SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 1, pp. 1-32, January 1990

A METHOD FOR THE SPATIAL DISCRETIZATION OF PARABOLIC EQUATIONS IN ONE SPACE VARIABLE*

ROBERT D. SKEEL† AND MARTIN BERZINS‡

Abstract. This paper is concerned with the design of a spatial discretization method for polar and nonpolar parabolic equations in one space variable. A new spatial discretization method suitable for use in a library program is derived. The relationship to other methods is explored. Truncation error analysis and numerical examples are used to illustrate the accuracy of the new algorithm and to compare it with other recent codes.

Key words. Galerkin method, Petrov-Galerkin method, polar coordinates, singular boundary value problems, parabolic equations, method of lines

AMS(MOS) subject classifications. 65M05, 65M10, 65M15, 65M20

C. R. classification. G.1.8

1. Introduction. The aim of this paper is to describe and to give evidence in support of a new spatial discretization for the method of lines solution of parabolic equations in one space variable. The intent is to provide a method that is suitable for use in a general-purpose library program, such as the D03P section of the NAG library.

Ordinary and parabolic partial differential equations in one space variable x often have a singularity due to the use of polar cylindrical or spherical coordinates. Because of their common occurrence, some of the differential equation software—such as the PDEONE code of Sincovec and Madsen [16] and the D03P** code of Dew and Walsh [8]—treat these singularities explicitly in order to reduce accuracy problems that arise for coefficients like $1/x^m$ when x is near zero. Nonetheless, methods that have been proposed (see Eriksson and Thomée [9] for references) or implemented do not obtain the same (local order of) accuracy for the case $m=1$ (and sometimes $m=2$) as they do for $m=0$.

The method we propose is a simple piecewise nonlinear Galerkin/Petrov-Galerkin method that is second-order accurate in space. (It supersedes the method proposed by Skeel [17].) The case $m=1$ involves the use of the logarithm function, which is probably the only accurate way to model the logarithmic behavior that can be present in the solution. A code based on a variant of the proposed method has already been included as part of the SPRINT package of Berzins, Dew, and Furzeland [4] (which is available from M. Berzins). The method we propose here has been implemented and will be distributed in the next or next but one release of the D03P (parabolic equations) section of the NAG library.

A derivation of the method is given in § 2. Rather than simply announcing our choice of trial space, test space, and inner product, we give a more concrete finite-element derivation that motivates these choices. Section 3 gives the concise Galerkin

* Received by the editors January 28, 1987; accepted for publication (in revised form) October 18, 1988.

† Department of Computer Science, University of Illinois at Urbana-Champaign, 240 Digital Computer Laboratory, 1304 West Springfield Avenue, Urbana, Illinois 61801. The research of this author was supported in part by the U.K. Science Research Council and in part by the U.S. Department of Energy under grant DEAC0276ER02383.

‡ Department of Computer Studies, University of Leeds, Leeds LS2 9JT, United Kingdom. The research of this author was partly supported by Shell Research Ltd., Thornton Research Centre, P.O. Box 1, Chester, United Kingdom.

formulation of the method. The primary purpose of the remainder of the paper is to supply the details of the evidence in favor of the new method. Section 4 discusses the variant of the proposed method used in SPRINT as well as other competing methods, § 5 considers the time integrator for the system of ordinary differential equations, and § 6 describes the results of numerical testing. Finally, § 7 summarizes our conclusions. Also, there is an Appendix that contains the error analysis that supports the claim of good accuracy for the proposed methods.

2. Derivation of the spatial discretization method. Consider the system of quasilinear partial differential equations (PDEs):

$$D(x, t, u, u_x)u_t = x^{-m}(x^m g(x, t, u, u_x))_x + f(x, t, u, u_x) \tag{1}$$

for $a \leqq x \leqq b$ where D is a diagonal matrix with nonnegative entries and m is nonnegative. If $m > 0$, we require $a \geqq 0$. The cases $m = 1$ and $m = 2$ represent cylindrical and spherical polar coordinates, respectively. Boundary conditions are

$$p^i(x, t, u) + q^i(x, t)g^i(x, t, u, u_x) = 0 \quad \text{at } x = a, b$$

for $i = 1, 2, \cdots$, NPDE. If $m \geqq 1$ and $a = 0$, the boundedness of the solution near $x = 0$ implies $g^i(x, t, u, u_x) = 0$ at $x = 0$. The problem class defined by (1) has been deliberately chosen so as to have recognizable flux and source terms and to have the possibility of recognizable Cartesian polar and spherical polar coordinates. The general form of the flux function $g(\cdots)$ is designed to permit conservative differencing of both advective and diffusive flux terms. For notational convenience we work with a single PDE and omit the argument t.

We consider a spatial mesh $a = x_0 < x_1 < \cdots < x_J = b$. Because continuity of the solution $u(x)$ and of the (negative) flux (per unit area) $v(x) := g(x, u(x), u_x(x))$ is demanded for all x, we use as unknowns values of u and v at meshpoints. It is assumed that meshpoints are placed at discontinuities in the PDE so that the problem is smooth within each subinterval.

The SPRINT implementation also has two additional features of interest. The first is that the problem class is extended to include the coupled ODE (ordinary differential equation)/PDE problems considered by Schryer [15]. The only restrictions are that the problem class must be linear in the PDE time derivative and that time derivatives must not appear in the flux term. The second feature is that an optional remeshing facility has been provided, based on the padded monitor function of Kautsky and Nichols [11]. The monitor function is chosen by the user and typically depends on the flux or on the solution and its space derivatives. These two features are discussed further by Berzins and Furzeland [5], [6].

We seek a difference scheme that

(i) Uses only one evaluation of D, f and g per subinterval;

(ii) Is elegant;

(iii) Is as accurate as possible for the special case

$$g(x, u, u_x) = G(x, u)u_x;$$

and

(iv) Leads to an explicit system of ODEs (which is desirable for reasons given in § 5). Ideally *local* second-order accuracy is desired, meaning that the contribution to the global error from subinterval $[x_{j-1}, x_j]$ is $O(h_j^3)$ where $h_j := x_j - x_{j-1}$. It is clear what this means for linear problems where the global error is a superposition of propagated local errors; more care would be required to define this idea for nonlinear problems. Global second-order accuracy is a weaker property meaning that the local errors are $O(h^3)$ on the average.

The special problem we consider can be expressed as the two first-order PDEs:

$$u_x = H(x, u)v, \tag{2}$$

$$(x^m v)_x = x^m Q(x, u, u_x, u_t) \tag{3}$$

where

$$H(x, u) := 1/G(x, u)$$

and

$$Q(x, u, u_x, u_t) := D(x, u, u_x)u_t - f(x, u, u_x).$$

The derivation begins by focusing on a typical subinterval, or element, that we denote by $[\alpha, \beta]$. The length $h := \beta - \alpha$ and the midpoint $\gamma := (\alpha + \beta)/2$. The first section considers quadrature and lumping, the second treats interpolation, and the third assembles the element equations into difference equations by eliminating the unknown values of $v(x)$ at meshpoints.

Two separate cases are considered depending on whether or not the $1/x^m$ singularity is present. The "regular case" occurs when $m = 0$ or $a > 0$ and is treated with a Galerkin method. The "singular case" occurs when $m \geqq 1$ and $a = 0$, and a Petrov-Galerkin method is used, in which a special trial space is chosen. In the singular case the presence of a discontinuity has an important effect, and for this reason the error analysis takes into account the location of the first discontinuity from the left, which we denote by c (so that on $[0, c]$ the problem is smooth). In the regular case when $m \geqq 1$ let $c := a$. The factor $1/c$ shows up in some error bounds, but we try to avoid it as much as possible in the belief that there are problems for which $c \ll b$ (such as an annulus with a very small center hole).

The choice of scheme was motivated by an error analysis, which in turn was influenced by numerical experiments. The detailed analysis, given in the Appendix, shows that the proposed scheme is second-order accurate for both the regular and singular cases outlined above. We obtain error bounds depending on problem parameters P consisting of bounds on various derivatives on the *open subintervals* of $[a, b]$. For the singular case we also use $\bar{P}$, which consists of bounds on the same derivatives on *all* of $[0, c]$, including interior meshpoints. Most of the error analysis for $m \geqq 1$ applies to any real $m \geqq 1$ not just $m = 1$ and $m = 2$.

2.1. Quadrature and lumping. Integrating (3) and then (2), we obtain

$$u(\beta) = u(\alpha) + \int_\alpha^\beta \frac{H(\cdots)}{x^m} \left\{ \alpha^m v(\alpha) + \int_\alpha^x y^m Q(\cdots)\, dy \right\} dx, \tag{4}$$

and interchanging α and β, we obtain

$$u(\alpha) = u(\beta) - \int_\alpha^\beta \frac{H(\cdots)}{x^m} \left\{ \beta^m v(\beta) - \int_x^\beta y^m Q(\cdots)\, dy \right\} dx. \tag{5}$$

The basic idea is that of the finite-element method. We form discrete approximations of these two equations, thus obtaining an equation for each of $v(\alpha)$ and $v(\beta)$ in terms of $u(\alpha)$ and $u(\beta)$. Then the boundary conditions and the continuity of $v(x)$ at meshpoints can be used to eliminate these unknown values of $v(x)$, thus obtaining difference equations for the values of $u(x)$ at meshpoints.

Approximations to (4) and (5) are developed by using the values

$$H_0 := H(\xi, u(\xi)), \qquad D_0 := D(\xi, u(\xi), u_x(\xi)),$$

$$f_0 := f(\xi, u(\xi), u_x(\xi))$$

at a point ξ inside the interval $[\alpha, \beta]$. (Error analysis suggests to us that separate choices of ξ for each of H, D, and f would, at best, offer only slight improvement in the accuracy of the scheme. Moreover, this would complicate the user interface, requiring either the provision of more than one PDE routine or the evaluation of different PDE functions at different space points in one call.) The question of approximating $u(\xi)$ and $u_x(\xi)$ is deferred to § 2.2. If we assume that with one degree of freedom H, D, and f are best approximated by constants, then we get

$$u(\beta)=u(\alpha)+\int_\alpha^\beta \frac{H_0}{x^m}\left\{\alpha^m v(\alpha)+\int_\alpha^x y^m(D_0u_t(\alpha)-f_0)\,dy\right\}dx+\sigma+H_0\tau_\alpha \tag{6}$$

and

$$u(\alpha)=u(\beta)-\int_\alpha^\beta \frac{H_0}{x^m}\left\{\beta^m v(\beta)-\int_x^\beta y^m(D_0u_t(\beta)-f_0)\,dy\right\}dx-\sigma+H_0\tau_\beta \tag{7}$$

where we have used $u_t(\alpha)$ in (6) and $u_t(\beta)$ in (7) to get an explicit system of ODEs in time and the quadrature errors σ, τ_α, and τ_β are defined by

$$\int_\alpha^\beta H(\cdots)v(x)\,dx=:H_0\int_\alpha^\beta v(x)\,dx+\sigma, \tag{8}$$

$$\int_\alpha^\beta \frac{1}{x^m}\int_\alpha^x y^mQ(\cdots)\,dy\,dx=:\int_\alpha^\beta\frac{1}{x^m}\int_\alpha^x y^m(D_0u_t(\alpha)-f_0)\,dy\,dx+\tau_\alpha, \tag{9}$$

and

$$\int_\alpha^\beta \frac{1}{x^m}\int_x^\beta y^mQ(\cdots)\,dy\,dx=:\int_\alpha^\beta\frac{1}{x^m}\int_x^\beta y^m(D_0u_t(\beta)-f_0)\,dy\,dx+\tau_\beta. \tag{10}$$

Note that (7) is undefined for $\alpha=0$ and $m\geqq 1$. For this and other reasons we develop an additional equation and its discretization. Combining (4) and (5), we obtain

$$\beta^m v(\beta)=\alpha^m v(\alpha)+\int_\alpha^\beta x^mQ(\cdots)\,dx, \tag{11}$$

and combining (6) and (7), we obtain

$$\beta^m v(\beta)=:\alpha^m v(\alpha)+\int_\alpha^\beta x^m\{D_0(u_t(\alpha)\psi_\alpha(x)+u_t(\beta)\psi_\beta(x))-f_0\}\,dx+\tau \tag{12}$$

where

$$\psi_\beta(x):=\begin{cases}\int_\alpha^x y^{-m}\,dy\Big/\int_\alpha^\beta y^{-m}\,dy, & \alpha>0 \text{ or } m=0,\\ 1, & \alpha=0 \text{ and } m\geqq 1,\end{cases}$$

and

$$\psi_\alpha(x):=1-\psi_\beta(x).$$

In the special case $\alpha=0$ and $m\geqq 1$, we use (12) instead of (7). Note that, otherwise, the quadrature error τ satisfies

$$\tau:=\frac{\tau_\alpha+\tau_\beta}{\int_\alpha^\beta y^{-m}\,dy}.$$

2.1.1. Choice of quadrature point. The choice of ξ gives us a second degree of freedom in the numerical quadrature of (6) and (7). We choose this point to obtain the most accuracy under the assumption that with two degrees of freedom H, D, and f are best approximated by linear polynomials.

The choice of ξ should take into account the behavior of $v(x)$. Consideration of simple examples, such as $H \equiv 1$ and $Q \equiv 1$, suggests that $v(x)$ behaves like x^{-m} in the regular case and like x in the singular case. Hence, we choose

$$\xi = \gamma_{-\mu}$$

where

$$\mu := \begin{cases} m & \text{regular case,} \\ -1 & \text{singular case,} \end{cases}$$

and where γ_k denotes the Gauss point for the weight function x^k, that is,

$$\int_\alpha^\beta (x-\gamma_k)x^k\,dx := 0.$$

Note that

$$\gamma_k = \gamma + \frac{k}{12}\frac{h^2}{\gamma} + O\left(\frac{h^4}{\gamma^3}\right)$$

and in particular $\gamma_1 = \gamma + h^2/(12\gamma)$ exactly.

This choice of quadrature point is justified in the Appendix where in Theorem 5 it is shown that with $\xi = \gamma_{-m}$ in the regular case the error

$$|\sigma| \leqq h^3 C(P)$$

where C denotes a generic constant (meaning that each occurrence of the symbol "C" is a possibly different symbol), and after this it is shown that other choices of ξ are theoretically inferior if a is small but positive. In Theorem 6 it is shown for the singular case that with $\xi = \gamma_1$ the error

$$|\sigma| \leqq \gamma h^3 C(\bar{P})/(m+1),$$

and after this it is shown that this bound is not possible if $\xi = \gamma$. The additional factor of γ in the error bound indicates an extra order of accuracy near the origin $x = 0$. Recall that because discontinuities in H, D, f and the initial conditions are permitted at meshpoints, the bound is good only on the interval $[0, c]$. On the remainder of the interval, $[c, b]$, Theorem 6 states that for the singular case with $\xi = \gamma_1$ the error

$$|\sigma| \leqq h^3 C(P),$$

which is no worse than the bound for $\xi = \gamma_{-m}$. Finally, it is not possible to choose ξ to be the same for both regular and singular cases because $\gamma_{-m} < \gamma < \gamma_1$.

The effect of τ_α and τ_β on the global error is not straightforward. This is discussed further in the Appendix. It is shown there that the error τ in (12) as an approximation to (11) is important but not so important so as to dictate the precise choice of ξ. However, in the singular case when $m > 1$, error analysis and numerical experiments show that the use of $\xi = \gamma_1$ is better than either the midpoint γ or the point γ_m suggested previously by Bakker [1].

2.2. Interpolation. Each of the functions $G(\cdots)$, $D(\cdots)$, and $f(\cdots)$ will have to be evaluated at ξ; and since our derivation assumes the availability of $u(x)$ only at

$x=\alpha$ and $x=\beta$, it will be necessary to construct an interpolant. Note that $u_x = Hv$ and recall that $v(x)$ behaves like x^{-m} in the regular case and like x in the singular case. This suggests for $\alpha \leqq x \leqq \beta$ the use of

$$U(x) := \begin{cases} u(\alpha)\psi_\alpha(x) + u(\beta)\psi_\beta(x) & \text{regular case,} \\ u(\alpha)\phi_\alpha(x) + u(\beta)\phi_\beta(x) & \text{singular case} \end{cases}$$

where ψ_α and ψ_β are defined in § 2.1 and where

$$\phi_\beta(x) := (x^2-\alpha^2)/(\beta^2-\alpha^2), \qquad \phi_\alpha(x) := 1-\phi_\beta(x).$$

The regular case interpolant has been suggested by Russell and Shampine [14] for collocation. Second-order accuracy for the regular case interpolant is shown in Theorem 1 and for the singular case interpolant in Theorem 3.

The derivative $U_x(\xi)$ is also used. Theorem 2 states that for the ψ interpolant

$$|u_x(\xi) - U_x(\xi)| \leqq \left(\frac{\zeta}{\xi}\right)^m \left(1+\frac{m}{\gamma}\right) h^2 C(P)$$

where

$$\zeta^{m+1} := \int_\alpha^\beta x\,dx \bigg/ \int_\alpha^\beta x^{-m}\,dx.$$

The point ζ is between γ_{-m} and γ, and so we have second-order accuracy for the regular case. Linear interpolation would cause this error to be increased by a factor of $1/\xi$ if $m \geqq 1$ and $a > 0$. Theorem 4 states that for the singular case with the ϕ interpolant

$$|u_x(\xi) - U_x(\xi)| \leqq \gamma h^2 C(\bar{P})/(m+1).$$

Again linear interpolation increases this error by a factor $1/\xi$. As before this bound for the singular case holds only on that interval $[0, c]$ on which Q is smooth. On the remainder of the interval Theorem 4 gives the bound

$$|u_x(\xi) - U_x(\xi)| \leqq \frac{m+1}{\gamma} h^2 C(P).$$

Having constructed an accurate interpolant, we replace $1/H_0$, D_0, and f_0 by $\tilde{G}_0 := G(\xi, U(\xi))$, $\tilde{D}_0 := D(\xi, U(\xi), U_x(\xi))$, and $\tilde{f}_0 := f(\xi, U(\xi), U_x(\xi))$ in (6), (7), and (12) and leave the precise form of the truncation errors for the Appendix. Equations (6) and (7) can be put into the simpler form:

$$\zeta^m \left(\frac{\xi}{\zeta}\right)^\mu \tilde{G}_0 U_x(\xi) = \alpha^m v(\alpha) + \frac{\zeta^{m+1} - \alpha^{m+1}}{m+1} (\tilde{D}_0 u_t(\alpha) - \tilde{f}_0) + \text{error} \tag{13}$$

and

$$\zeta^m \left(\frac{\xi}{\zeta}\right)^\mu \tilde{G}_0 U_x(\xi) = \beta^m v(\beta) - \frac{\beta^{m+1} - \zeta^{m+1}}{m+1} (\tilde{D}_0 u_t(\beta) - \tilde{f}_0) + \text{error.} \tag{14}$$

The foregoing derivation does not quite work for the special case $\alpha = 0$ and $m \geqq 1$. Because $\zeta = 0$, we do not want to multiply (6) by ζ^{m+1}. Therefore in (13) we adopt the understanding that we first divide by ζ^{m+1} with $\alpha > 0$ and then take the limit $\alpha \to 0$. (Note that $\alpha^m/\zeta^{m+1} \to 0$.) Also, because (7) is undefined, we use (12) to derive (14).

2.3. Assembly of the equations. If the term $\tilde{G}_0 U_x(\xi)$ in (13) and (14) is generalized to $g(\xi, U(\xi), U_x(\xi))$ and the truncation errors are neglected, we get difference equations for the semidiscrete solution $\{u_j\}$, $\{v_j\}$. If we identify $[\alpha, \beta]$ with $[x_{j-1}, x_j]$, then with an obvious change of notation we get

$$(15_j) \qquad \zeta_{j-1/2}^{m-\mu}\xi_{j-1/2}^{\mu}g_{j-1/2} = x_{j-1}^m v_{j-1} + \frac{\zeta_{j-1/2}^{m+1} - x_{j-1}^{m+1}}{m+1}(D_{j-1/2}\dot{u}_{j-1} - f_{j-1/2}),$$

and

$$(16_j) \qquad -\zeta_{j-1/2}^{m-\mu}\xi_{j-1/2}^{\mu}g_{j-1/2} = -x_j^m v_j + \frac{x_j^{m+1} - \zeta_{j-1/2}^{m+1}}{m+1}(D_{j-1/2}\dot{u}_j - f_{j-1/2}),$$

for $j = 1, 2, \cdots, J$. At interior meshpoints x_j we add (15_{j+1}) to (16_j) to obtain

$$\zeta_{j+1/2}^{m-\mu}\xi_{j+1/2}^{\mu}g_{j+1/2} - \zeta_{j-1/2}^{m-\mu}\xi_{j-1/2}^{\mu}g_{j-1/2}$$
$$= \frac{\zeta_{j+1/2}^{m+1} - x_j^{m+1}}{m+1}(D_{j+1/2}\dot{u}_j - f_{j+1/2}) + \frac{x_j^{m+1} - \zeta_{j-1/2}^{m+1}}{m+1}(D_{j-1/2}\dot{u}_j - f_{j-1/2}).$$

At the right boundary we can solve (16_J) for v_J and substitute this for $g(b, u(b), u_x(b))$ in the right boundary condition. The same approach is used at the left boundary unless $m \geqq 1$ and $a = 0$ in which case we divide (15_1) by $\zeta_{1/2}^{m+1}$ with $a > 0$ and then take the limit $a \to 0$.

3. Galerkin formulation. Integration of the PDE (1) on $[\alpha, \beta]$ with weight function x^m and test function $\psi(x)$ and then integration by parts yields

$$(17) \qquad \psi(\beta)\beta^m v(\beta) - \psi(\alpha)\alpha^m v(\alpha) - \int_\alpha^\beta \psi_x g x^m\,dx = \int_\alpha^\beta \psi Q x^m\,dx.$$

We use as our shape functions $\psi_\alpha(x)$ and $\psi_\beta(x)$ in the regular case and $\phi_\alpha(x)$ and $\phi_\beta(x)$ in the singular case, and so our interpolant $U(x)$ is as defined in § 2.2. With $\psi = \psi_\alpha$, (17) becomes

$$-\int_\alpha^\beta x^m g\psi_{\alpha x}\,dx = \alpha^m v(\alpha) + \int_\alpha^\beta Q x^m \psi_\alpha\,dx,$$

and after numerical quadrature and lumping we get

$$-\xi^\mu g(\xi, U(\xi), U_x(\xi)) \int_\alpha^\beta x^{m-\mu}\psi_{\alpha x}\,dx$$
$$= \alpha^m v(\alpha) + Q(\xi, U(\xi), U_x(\xi), U_t(\alpha)) \int_\alpha^\beta x^m \psi_\alpha\,dx,$$

which is the same as (15_j). In a similar way as with $\psi = \psi_\beta$ we get (16_j).

In the *regular* case the test and trial (shape) functions are the same and thus the method is of *Galerkin* type. In the *singular* case the test and trial functions are different and thus the method is of *Petrov-Galerkin* type.

4. Other methods. In this section we discuss other low-order methods for solving the problem under consideration. The method described in § 4.1 is nearly the same as the method proposed in §§ 2 and 3, and it has been implemented in the SPRINT package. The remaining sections discuss linear Galerkin methods of Bakker [1], Eriksson and Thomée [9], and Berzins and Dew [3] and finite difference methods used in the algorithm PDEONE [15] and in the D03P** family [8] of NAG library routines.

All of these other methods, including the one implemented in SPRINT, use linear interpolation for $U(x)$. As explained in the Appendix this makes no difference to the order of the truncation error and matters only if a is small but positive.

4.1. The method used in SPRINT. The method implemented in SPRINT software of Berzins, Dew, and Furzeland [4] was developed as a first stage in the eventual development of the method described in § 2. It is a Petrov-Galerkin method that includes only some of the features of the proposed Galerkin method that improve the order of the local truncation error. In the nonpolar regular case the method is identical to that of § 2.

The test functions for an element $[\alpha, \beta]$ are chosen to be

$$\psi_\alpha(x) := \int_x^\beta y^{-m}\,dy \bigg/ \int_\alpha^\beta y^{-m}\,dy$$

except if $m > 0$ and $\alpha = 0$, in which case

$$\psi_\alpha(x) = (\beta - x)/(\beta - \alpha)$$

and in both cases

$$\psi_\beta(x) := 1 - \psi_\alpha(x).$$

Instead of evaluating $x^m g$ at $x = \xi$ we evaluate g at $x = \gamma$, and the evaluation of D and f is at γ rather than ξ except for the case $m = 1$ where it is necessary to use ξ to maintain a propagated local error of $O(h^3)$.

4.2. The method used in PDEF1. A piecewise linear Galerkin method has been implemented by Bakker [1]. The crucial difference from the method described in § 2 is this: instead of (8) we get

$$\int_\alpha^\beta H(\cdots)v(x)\,dx = \frac{hH(\gamma, U(\gamma))}{\int_\alpha^\beta x^m\,dx} \int_\alpha^\beta x^m v(x)\,dx + \bar{\sigma}.$$

If we consider the example $H \equiv 1$ and $v(x) = x$, we have

$$\bar{\sigma} = -h(\gamma_m - \gamma) = -\frac{m}{12}\frac{h^3}{\gamma} + O\left(\frac{h^5}{\gamma^3}\right)$$

so that one order of accuracy is lost near the origin. This is serious because the accumulation of such local errors leads to a global error of $O(h^2 \log(1/h))$, an estimate derived by Jespersen [10].

4.3. The method of Eriksson and Thomée. To obtain better accuracy when $m > 1$, Eriksson and Thomée [9] consider the more specialized PDE

$$x^{-1}(xu_x + (m-1)u)_x = Q(x, u, u_x, u_t).$$

A variational equation is obtained by replacing u with its piecewise linear interpolant U, multiplying by a "hat" function $\phi_j(x)$, and integrating with weight function x. The stiffness matrix, in the case $Q(\cdots) = q(x)u - f(x)$, is not symmetric for this method, although it is for the method of § 4.1. However, it is proved [9] that the global error is $O(h^2)$.

Before applying numerical quadrature we obtain the equations

$$\left(\alpha + \frac{mh}{2}\right)\frac{u(\beta) - u(\alpha)}{h} = \alpha v(\alpha) + \int_\alpha^\beta x\phi_\alpha(x)Q(x, U(x), U_x(x), U_t(x))\,dx + \text{error}$$

and

$$-\left(\beta-\frac{mh}{2}\right)\frac{u(\beta)-u(\alpha)}{h}=-\beta v(\beta)+\int_\alpha^\beta x\phi_\beta(x)Q(x, U(x), U_x(x), U_t(x))\,dx+\text{error}.$$

In the case $m=2$ this is identical to the method of § 4.1 if the Q's are replaced by their lumped midpoint values and U by a linear interpolant (and it is identical to the finite difference method of Chawla and Katti [7] if the trapezoidal rule is used for quadrature). If $m\neq 2$, the methods are quite different.

4.4. The method used in SGENCO. If the Berzins and Dew [3] C^0 collocation code SGENCO is used with linear basis functions for the equation

$$(g(x, u, u_x))_x=Q(x, u, u_x, u_t)-\frac{m}{x}g(x, u, u_x),$$

then we get the equations

$$\frac{\alpha+mh}{2}g_{\alpha+}+\frac{\alpha}{2}g_{\beta-}=\alpha v(\alpha)+\frac{h}{2}\alpha Q_{\alpha+}+\text{error}$$

and

$$-\frac{\beta}{2}g_{\alpha+}-\frac{\beta-mh}{2}g_{\beta-}=-\beta v(\beta)+\frac{h}{2}\beta Q_{\beta-}+\text{error}$$

where

$$Q_{\alpha+}=Q(\alpha+, u(\alpha), U_x(\alpha+), u_t(\alpha)),\quad \text{etc.}$$

If $g(x, u, u_x)\equiv u_x$, then this is identical to the method of Eriksson and Thomée with trapezoidal quadrature. For $m=1$ it is thus a Galerkin method with weight function x, and the global error is $O(h^2\log(1/h))$.

4.5. The method used in PDEONE. We derive here a scheme like that of Sincovec and Madsen [16] and Varga [18, p. 175]. In (4) and (5) instead of evaluating H and Q at $x=\xi$, we evaluate the entire integrand at $x=\gamma$ giving

$$u(\beta)=u(\alpha)+h\frac{H_0}{\gamma^m}\left\{\alpha^m v(\alpha)+\int_\alpha^\gamma x^m Q(\cdots)\,dx\right\}+\bar\sigma$$

and

$$u(\alpha)=u(\beta)-h\frac{H_0}{\gamma^m}\left\{\beta^m v(\beta)-\int_\gamma^\beta x^m Q(\cdots)\,dx\right\}-\bar\sigma$$

where $H_0=H(\gamma, U(\gamma))$. For the integrals of $Q(\cdots)$ we use one-point quadrature rules yielding

$$u(\beta)=u(\alpha)+h\frac{H_0}{\gamma^m}\left\{\alpha^m v(\alpha)+\frac{\gamma^{m+1}-\alpha^{m+1}}{m+1}Q_{\alpha+}\right\}+H_0\bar\tau_\alpha+\bar\sigma \tag{18}$$

and

$$u(\alpha)=u(\beta)-h\frac{H_0}{\gamma^m}\left\{\beta^m v(\beta)-\frac{\beta^{m+1}-\gamma^{m+1}}{m+1}Q_{\beta-}\right\}+H_0\bar\tau_\beta-\bar\sigma. \tag{19}$$

If we proceed as in § 2 we obtain the difference equation

$$\begin{aligned}(20)\quad &\frac{m+1}{x_{j+1/2}^{m+1}-x_{j-1/2}^{m+1}}(x_{j+1/2}^m g_{j+1/2}-x_{j-1/2}^m g_{j-1/2})\\ &=\frac{x_{j+1/2}^{m+1}-x_j^{m+1}}{x_{j+1/2}^{m+1}-x_{j-1/2}^{m+1}}Q_{j+}+\frac{x_j^{m+1}-x_{j-1/2}^{m+1}}{x_{j+1/2}^{m+1}-x_{j-1/2}^{m+1}}Q_{j-}.\end{aligned}$$

Remark. The averaging of Q_{j+} and Q_{j-} actually recommended by Sincovec and Madsen [16, p. 242] is different; it is obtained by setting $m=0$ in the right-hand side of (20). Also the spatial derivative term in $Q(x, u, u_x, u_t)$ is approximated by $(u_{j+1}-u_{j-1})/(h_{j+1}+h_j)$, which is not so accurate if u_x has a discontinuity at x_j.

Combining (18) and (19), we get

$$\beta^m v(\beta)=\alpha^m v(\alpha)+\frac{\beta^{m+1}-\gamma^{m+1}}{m+1}Q_{\beta-}+\frac{\gamma^{m+1}-\alpha^{m+1}}{m+1}Q_{\alpha+}+\bar{\tau}$$

where

$$\bar{\tau}:=\int_\alpha^\beta x^m Q(\cdots)\,dx-\frac{\beta^{m+1}-\gamma^{m+1}}{m+1}Q_{\beta-}-\frac{\gamma^{m+1}-\alpha^{m+1}}{m+1}Q_{\alpha+}.$$

If we consider $H\equiv 1$, $Q(x)=x$, and $\alpha=0$, then

$$\begin{aligned}\bar{\tau}_S&=\int_h^1\frac{dy}{y^m}\cdot\bar{\tau}+\bar{\tau}_\alpha\\ &=\int_h^1\frac{dy}{y^m}\cdot\frac{m+2-2^{m+1}}{(m+1)(m+2)2^{m+1}}h^{m+2}+\frac{h^3}{4(m+2)},\end{aligned}$$

and if $m=1$ the resulting contribution to the global error is $O(h^3\log(1/h))$. Nonetheless the accumulation of such errors is only $O(h^2)$.

The error $\bar{\sigma}$ is simply

$$\bar{\sigma}=\int_\alpha^\beta u_x(x)-u_x(\gamma)\,dx.$$

If $a>0$, $H\equiv 1$, $Q\equiv 0$, $v(a)=1$, and $\alpha=a$, then

$$\bar{\sigma}=\frac{m(m+1)}{24}\frac{h^3}{a^2}+O(h^4),$$

which is worse than the method of § 4.1 by a factor of $1/a$ and worse than the method of § 2 by a factor of $1/a^2$. If we consider $a=0$, $H\equiv 1$, $Q=0$ for $x<c$, $Q=1$ for $x>0$, and $\alpha=c$, then

$$\bar{\sigma}=-\frac{m}{24}\frac{h^3}{c}+O(h^4),$$

which is worse than the methods of §§ 2 and 4.1 by a factor of $1/c$.

In conclusion we see that this method does achieve global second-order accuracy for general n but is less accurate than the method proposed in § 2.

4.6. The method used in D03P.** The difference scheme of this collection of NAG routines by Dew and Walsh [8] is modeled after that of PDEONE. To simplify somewhat the usage of the routines, the coefficient $G(\gamma, U(\gamma))$ is replaced by $\frac{1}{2}(G(\alpha, U(\alpha))+G(\beta, U(\beta)))$. This is all right except at a discontinuity x_j where there seems to be no way to define an averaging for $G(x_j, u_j)$ that does not degrade the accuracy in either the left subinterval or right subinterval. The codes D03P** and PDEONE are compared by Berzins and Dew [2].

5. Integration in time. The issue of integration in time is not considered in any detail here. Instead it is noted that the spatial discretization of elliptic-parabolic PDEs using the method of § 2 results in large systems of differential-algebraic equations that are integrated using standard software (see Berzins, Dew, and Furzeland [4] and Petzold [12] for further details).

5.1. Explicit or implicit ODEs. Although it is possible to integrate stiff implicit ODEs with almost the same overhead as explicit ODE systems written in normal form, there are still a number of reasons why it is preferable to reduce systems to normal form whenever possible, such as by the lumping applied in § 2.1.

A substantial difficulty with implicit differential or differential-algebraic equations lies in the calculation of the initial solution values and their time derivatives (see Petzold [12]). This is not a problem with systems written in normal form. A further difficulty is that with implicit equations it is sometimes possible to calculate physically misleading values for the initial time derivatives. This point is easily illustrated by the simple example below and leads to a noticeable deterioration of the performance of the ODE integrator.

5.1.1. Example. Consider the heat equation $u_t = u_{xx}$ with boundary and initial conditions given by

$$u(0,t) = 0,$$
$$u(1,t) = \begin{cases} 0, & t < 1, \\ 1, & t \geqq 1, \end{cases}$$
$$u(x,0) = 0,$$

and suppose that we semidiscretize in space without lumping using a uniform mesh. This yields the system of ODEs

$$\frac{1}{6}\dot{u}_{j+1} + \frac{2}{3}\dot{u}_j + \frac{1}{6}\dot{u}_{j-1} = \frac{1}{h^2}(u_{j+1} - 2u_j + u_{j-1}).$$

The analytical solution (the limiting case of the backward Euler solution as the time stepsize goes to zero) at time $t = 1$ is

$$u_j(1) = \left(\frac{-1}{2+\sqrt{3}}\right)^{J-j} \frac{1-(2+\sqrt{3})^{-2j}}{1-(2+\sqrt{3})^{-2J}},$$

and in particular

$$u_{J-1}(1) \cong -.268, \quad u_{J-2}(1) \cong .072, \quad u_{J-3}(1) \cong -.019$$

for large J. Thus, the semidiscrete solution at interior meshpoints is discontinuous in time and oscillates in space with every other value having the wrong sign.

More generally, a discontinuity or a rapid variation of a boundary value produces corresponding behavior with alternating signs at nearby meshpoints. A discontinuity is often present at $t = 0$, when the boundary condition is inconsistent with the initial condition and the PDE. For example, if $u(1,t) = \sin t$ in the above example, then the derivatives $\dot{u}_J(0+) = 1$, $\dot{u}_{J-1}(0+) \cong -.268$, etc. We have observed that the effect of all this is to degrade the efficiency of the integration.

A further incentive to derive ODE systems in normal form is provided by a new generation of ODE initial value problem codes for both stiff and nonstiff equations (for example, see Petzold [13]). Such integrators attempt to use functional iteration whenever possible to increase the efficiency of integration. The unsuitability of functional iteration for the systems of equations that arise from implicit ODEs means that these codes cannot be applied to such equations at the moment.

6. Numerical testing on parabolic equations. In the numerical testing that was conducted the following measures were taken to ensure that a fair comparison was made. First, all integrations were performed using the same ODE integrator and the same linear algebra routines. The integrator used was the BDF/Adams code with the LINPACK banded matrix routines as implemented in SPRINT (Berzins, Dew, and Furzeland [4]). All discretization methods compared here were compared in a common framework. Only minor changes had to be made to the PDEONE code of Sincovec and Madsen [16] and to the PDEF1 code of Bakker [1] to fit them into this framework. These codes are abbreviated in this report as follows:

PDEONE: the Sincovec and Madsen [16] code,
SGENCO: the Berzins and Dew [3] C^0-collocation code used with linear basis functions,
SPRINT: the discretization method of § 4.1,
PDEF1: the lumped finite element method of Bakker [1] that uses linear basis functions,
NEW: the discretization method of § 2.

6.1. Nonpolar parabolic equations. Testing over a range of simple nonpolar parabolic equations has shown that the formula used by Bakker [1] and by Skeel [17] gives consistently better results than the finite difference methods of Dew and Walsh [8] and Sincovec and Madsen [16], although only to the same order of accuracy. The following problem gives results typical of those obtained on the nonpolar test problems of Berzins and Dew [3]. The methods of Skeel [17], §§ 4.1 and 2 are identical for nonpolar test problems.

6.1.1. Problem 1.1. This problem has an analytic solution and a material interface at $x=0$. The problem is defined by

$$\frac{\partial u}{\partial t}=\frac{\partial}{\partial x}\left(\frac{1}{C_1}\frac{\partial u}{\partial x}\right)+C_1e^{-2u}+e^{-u},\qquad x\in[-1,0),$$

$$\frac{\partial u}{\partial t}=\frac{\partial}{\partial x}\left(\frac{1}{C_2}\frac{\partial u}{\partial x}\right)+C_2e^{-2u}+e^{-u},\qquad x\in(0,1],$$

$$(x,t)\in[-1,1]\times(0,1]$$

subject to the boundary conditions

$$u(-1,t)=\log(-C_1+t+P),$$

$$u(1,t)+(C_2+t+P)\frac{\partial u}{\partial x}(1,t)=\log(C_2+t+P)+1.0.$$

The initial condition is consistent with the analytic solution of

$$u(x,t)=\log(C_ix+t+P)$$

where $i=1$ if $x<0$ and $i=2$ if $x>0$. In this case $P=1.1$, $C_1=0.1$, and $C_2=1.0$. This problem illustrates well the performance of the three codes on nonpolar parabolic

TABLE 1
Error norms for Problem 1.1.

Time	0.01	0.22	0.55	0.77	1.0
$N=11$					
PDEONE	1.9-2	9.0-3	3.9-3	2.7-3	1.9-3
SPRINT	1.9-2	8.3-3	3.7-3	2.3-3	1.5-3
SGENCO	1.4-2	6.4-3	2.8-3	1.9-3	1.3-3
$N=41$					
PDEONE	1.5-3	6.0-4	2.5-4	1.7-4	1.2-4
SPRINT	1.3-3	5.4-4	2.4-4	1.5-4	9.9-5
SGENCO	1.5-3	6.0-4	2.5-4	1.8-4	1.2-4
$N=161$					
PDEONE	9.4-5	3.8-5	1.6-5	1.1-5	7.6-6
SPRINT	7.8-5	3.4-5	1.5-5	9.3-6	6.2-6
SGENCO	9.4-5	3.8-5	1.6-5	1.1-5	7.6-6

TABLE 2
Maximum grid errors.

$N=$	11	21	41	81	161
PDEONE	1.9-2	6.2-3	1.7-3	4.3-3	1.1-4
SPRINT	1.3-2	3.3-3	8.3-4	2.1-4	5.2-5
SGENCO	1.9-2	6.1-3	1.7-3	4.3-3	1.1-4

equations. The Dew and Walsh [8] code cannot be compared as it does not correctly treat the material interface at $x=0$. The Bakker [1] code PDEF1 gives identical results to SPRINT for nonpolar problems. Table 1 shows the L^2 error norms at different time levels as the number of equally-spaced meshpoints is increased. N is the number of evenly-spaced meshpoints used in spatial discretization.

The error norms were formed by using a 201-point trapezoidal rule with evenly-spaced meshpoints and with solution values in between the PDE meshpoints being estimated by linear interpolation. Table 2 shows the maximum grid error for each of the methods sampled over the time values used in Table 1 with the additional points 0.11, 0.33, 0.44, 0.66, 0.88. The factor of two difference between the codes SPRINT and SGENCO can be directly attributed to the fact that both are essentially lumped Galerkin methods but that SPRINT uses a midpoint quadrature rule and SGENCO uses a trapezoidal rule.

6.2. Numerical testing on polar parabolic equations. The following test problems were used to compare the different codes on polar parabolic test problems.

6.2.1. Problem 2.1.

$$u\frac{\partial u}{\partial t}=\frac{1}{x^2}\frac{\partial}{\partial x}\left(x^2u\frac{\partial u}{\partial x}\right)+5u^2+4xu\frac{\partial u}{\partial x}, \qquad (x,t)\in[0,1]\times(0,1].$$

The left-hand boundary condition is the symmetry condition and the right-hand Dirichlet condition and the initial condition are consistent with the analytic solution of

$$u(x,t)=e^{1-x^2-t}.$$

6.2.2. Problem 2.2.

$$\frac{\partial u}{\partial t}=\frac{1}{x^2}\frac{\partial}{\partial x}\left(x^2\frac{1}{6}\frac{\partial u}{\partial x}\right)+\frac{2}{3}x^2e^{-2u}, \qquad (x,t)\in[0,1]\times(0,1].$$

The boundary conditions are the symmetry condition at $x=0$ and the boundary condition at $x=1$ given by

$$u(1,t)+(1.0+p+t)\frac{\partial u}{\partial x}=2.0+\log(1+p+t)$$

where $p=1.0$ and the analytic solution is given by

$$u(x,t)=\log(x^2+p+t).$$

6.2.3. Problem 2.3.

$$\frac{\partial u}{\partial t}=\frac{1}{x^2}\frac{\partial}{\partial x}\left(x^2\frac{\partial u}{\partial x}\right)+F(x,t), \qquad (x,t)\in[0,1]\times(0,1]$$

where

$$F(x,t)=e^{-t}[(6+(1-x^2)(\pi^2t^2-1))\cos(\pi xt)$$
$$-[(1-x^2)x+4xt-2t/x]\pi\sin(\pi xt)].$$

The boundary conditions are the symmetry condition at $x=0$ and $u=0$ at $x=1$. The exact solution is given by

$$u(x,t)=(1-x^2)e^{-t}\cos(\pi xt).$$

The limiting value of the function F at $x=0$ is obtained by using

$$\lim_{x\to 0}\frac{\sin(\pi xt)}{x}=\pi t.$$

6.2.4. Problem 2.4.

$$\frac{\partial u}{\partial t}=\frac{1}{x^2}\frac{\partial}{\partial x}\left(x^2\frac{\partial u}{\partial x}\right), \qquad (x,t)\in[0,1]\times(0,0.8]$$

where the boundary conditions are the symmetry condition at $x=0$ and

$$u(1,t)=1+6t.$$

The exact solution is given by

$$u(x,t)=x^2+6t.$$

6.2.5. Problem 2.5. This problem consists of the elliptic parabolic system defined by

$$\frac{\partial u}{\partial t}=\frac{1}{x}\frac{\partial}{\partial x}\left(x\frac{\partial u}{\partial x}\right)+F(x),$$

$$0=\frac{1}{x}\frac{\partial}{\partial x}\left(x\frac{\partial v}{\partial x}\right)+F(x),$$

$$(x,t)\in[0,1]\times(0,1]$$

where $F(x)=x$ for $x<p$ and equal to zero otherwise and $p=0.1$. The boundary conditions are the Neumann condition

$$\frac{\partial u}{\partial x}=\frac{\partial v}{\partial x}=0 \quad \text{at } x=0$$

and the Dirichlet condition

$$u=v=u_{\text{exact}} \quad \text{at } x=1$$

where the exact solution is given by

$$u=v=-\log(x)\frac{p^3}{3}, \qquad x>p$$

$$=-\log(p)\frac{p^3}{3}+\frac{p^3-x^3}{9}, \qquad x\leqq p.$$

6.2.6. Problem 2.6. The heat equation in cylindrical polar coordinates is

$$\frac{\partial u}{\partial t}=\frac{1}{x}\frac{\partial}{\partial x}\left(x\frac{\partial x}{\partial x}\right), \qquad (x,t)\in[0,1]\times(0,1]$$

where the exact solution is given by

$$u(x,t)=J_0(px)\,e^{-p^2t}$$

and $p\approx 2.40482557$ is the first zero of the Bessel function $J_0(x)$. The boundary conditions are a Dirichlet condition at $x=1$ and the usual symmetry condition at $x=0$.

6.2.7. Problem 2.7. The following problem is taken from Eriksson and Thomée [9]:

$$\frac{\partial u}{\partial t}=\frac{1}{x^2}\frac{\partial}{\partial x}(x^2u)-3u+\frac{\sinh(2x)}{x\sinh 2}-4e^t+3,$$

$$(x,t)\in[0,1]\times(0,1].$$

The boundary and initial conditions are given by

$$\frac{\partial u}{\partial x}(0,t)=u(1,t)=u(x,0)=0, \qquad t>0$$

and the exact solution is given by

$$u(x,t)=(e^t-1)\frac{\sinh(2x)}{x\sinh 2}-e^t+1.$$

6.2.8. Problem 2.8. The following is a slightly more complicated problem in cylindrical polar coordinates:

$$\frac{\partial u}{\partial t}=\frac{1}{x}\frac{\partial}{\partial x}\left(x\frac{\partial u}{\partial x}\right)+3u+2x\frac{\partial u}{\partial x},$$

$$(x,t)=[0,1]\times(0,1].$$

The boundary conditions are given by

$$\frac{\partial u}{\partial x}(0,t)=0, \quad u(1,t)=e^{-t}, \quad t>0$$

and the exact solution and the initial condition are given by

$$u(x, t) = e^{1-t-x^2}.$$

6.2.9. Problem 2.9. This problem consists of the parabolic equation defined by

$$\frac{\partial u}{\partial t} = \frac{1}{x}\frac{\partial}{\partial x}\left(x\frac{\partial u}{\partial x}\right) + F(x), \qquad (x, t) \in [0, 1] \times (0, 1]$$

where $F(x) = 100$ for $x < 0.1$ and equal to zero otherwise. The boundary conditions are the symmetry condition at $x = 0$

$$\frac{\partial u}{\partial x} = 0 \quad \text{at } x = 0$$

and the Dirichlet condition

$$u = 0 \quad \text{at } x = 1$$

where the exact solution is given by

$$u = \begin{cases} 0.5 \log(0.1) + 25((0.1)^2 - x^2) + J_0(px)\, e^{-p^2 t}, & x \leqq 0.1, \\ 0.5 \log x + J_0(px)\, e^{-p^2 t}, & x \geqq 0.1 \end{cases}$$

where $p \approx 2.40482557$ is the first zero of the Bessel function $J_0(x)$. This problem has a severe discontinuity in the PDE defining function close to the polar origin.

6.3. Summary of numerical testing results. The numerical testing results are summarized by the eight graphs of Fig. 1. The results for Problem 2.4 are not presented graphically and this problem is covered separately below. The procedure employed for each of the test problems was to use five evenly-spaced meshes of 11, 21, 41, 81, and 161 meshpoints. Each of the integrations in time was performed to a local ODE error tolerance that was sufficiently small for the PDE spatial discretization error to dominate the global error in the solution. All the graphs below are of the $\log_{10}$ of the maximum grid error against the $\log_{10}$ of the number of spatial meshpoints. The maximum error at the spatial meshpoints was found by stopping the integration at times $t = 0.01$ and then $t = k/9$ for $k = 1, 2, \cdots, 9$.

For the sake of clarity certain codes were not included on certain graphs. The following points should be noted:

The Bakker code PDEF1 is not applicable to Problem 2.1.

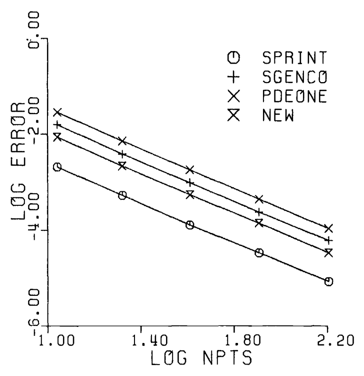
The results produced by PDEONE and SGENCO are indistinguishable on Problems 2.5, 2.6, 2.7, and 2.9.

The graph for Problem 2.6 compares the published results at $t = 1.0$ of Thomée and Eriksson [9] with the results at $t = 1.0$ for the other codes.

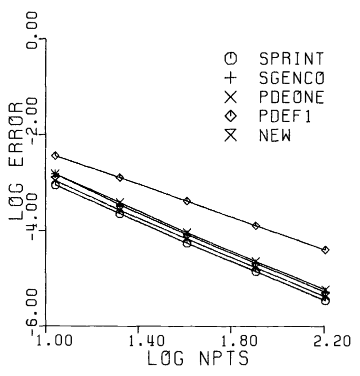
The codes SGENCO and PDEONE have a great deal of difficulty with the discontinuity in Problem 2.9.

The SPRINT code and the method of § 2 produce identical results for Problems 2.8, 2.6, and 2.9.

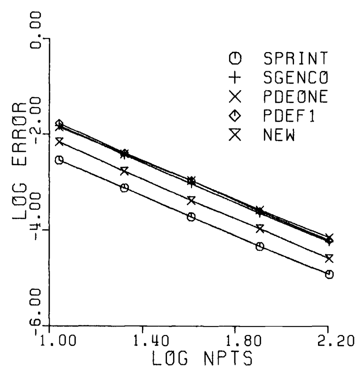
The results for Problem 2.4 are not presented graphically because all the codes apart from PDEF1 produce solutions that are exact at the meshpoints to computer roundoff error. The results for the maximum grid error at the meshpoints (EMAX) for code PDEF1 are as follows.



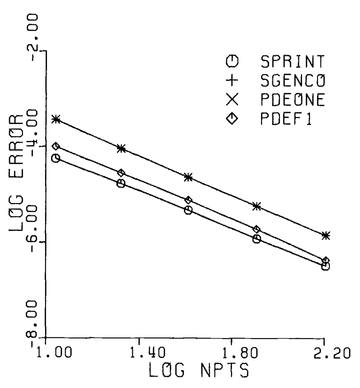
Problem 2.1



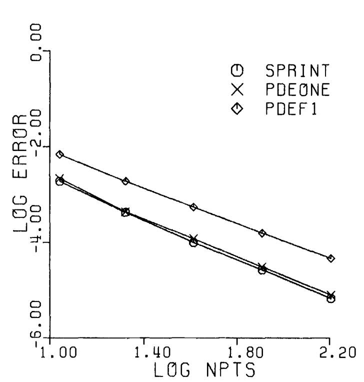
Problem 2.2



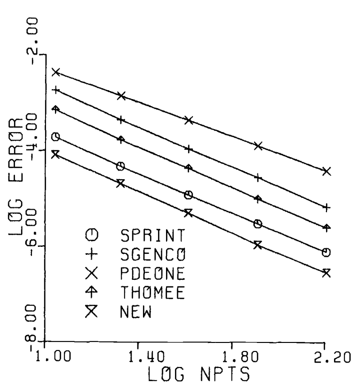
Problem 2.3



Problem 2.5

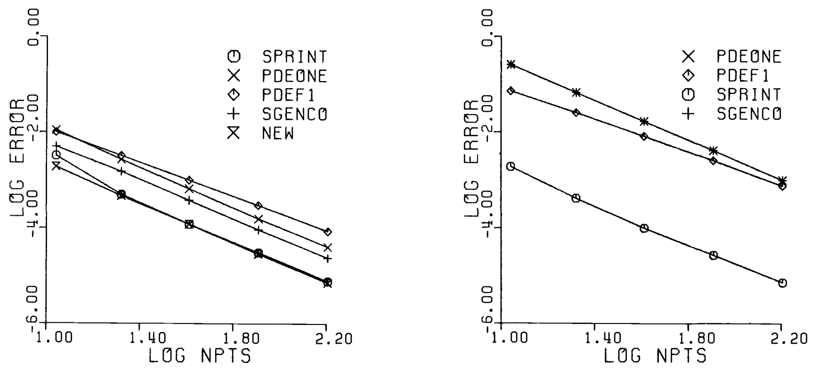


Problem 2.6



Problem 2.7

FIG. 1. *Summary of numerical testing results.*



Problem 2.8 Problem 2.9

FIG. 1—continued.

Max. Grid errors for PDEF1 on Prob. 2.4.

NPTS	11	21	41	81	161
EMAX	6.1E−1	2.1E−1	6.6E−2	2.0E−3	5.9E−3

The explanation for the poor showing of this code on this problem is that it semidiscretizes the PDE of Problem 2.4 to give the following equation at the internal meshpoints:

$$u_t = \frac{6 + h^2/2x^2}{1 + h^2/4x^2} \quad \text{at } x = x_j$$

where h is the even mesh spacing; whereas the other codes semidiscretize the PDE to the exact equation given by

$$u_t = 6 \quad \text{at } x = x_j.$$

The results for Problem 2.9 in particular show the advantage of the discretization method that we have developed over PDEONE and SGENCO.

The results for the Dew and Walsh [8] code D03PGF are not presented in the tables and graphs. On all the problems except Problems 2.2 and 2.5 the performance of the code is slightly worse than but very similar to SGENCO. On Problem 2.5 the code does not perform well due to the way it treats the discontinuities in the first derivative of the solution.

The graphical results clearly illustrate the superior performance of the SPRINT discretization and the discretization of § 2. The failure of the latter to perform the best on every problem has been analyzed, and it appears to be due to the fact that the new method has been designed to minimize a bound on the individual contribution of each local error rather than to achieve some cancellation of errors from different elements. Such cancellation can be fully realized, however, only for smooth problems on uniform meshes. Thus, it could be suggested that the method of § 2 sacrifices small possible gains in accuracy in order to achieve greater robustness.

7. Conclusions. Theoretical and experimental evidence indicates that the Galerkin method derived in this paper for the regular case and the Petrov-Galerkin method derived for the singular case produce more accurate results than existing methods and in particular they seem to be the best methods to use in a general-purpose library subroutine.

Appendix A. Error analysis. This Appendix gives bounds on the local truncation error of the method derived in § 2. Some of the proofs are longer than necessary in order to obtain tighter bounds, although this does not show up in the statements of the theorems. We use P to denote the 17-tuple consisting of the supremum norms on the open subintervals of $[a, b]$ of u_t, u_{tt}, v, v_t, H, H_x, H_t, H_{xx}, H_{xt}, D, D_x, D_t, D_{xx}, f, f_x, f_t, f_{xx}. Here H_x denotes $(H(x, u(x)))_x$ rather than $H_x(x, u)$ evaluated at $u = u(x)$. This is similar for the other derivatives of H, D, and f with respect to x and t. For the singular case we use $\bar{P}$ to denote a similar 17-tuple except that we use the supremum norm on all of $[0, c]$ for all 17 quantities (although for H_{xx} open subintervals would suffice). The error analysis assumes $b = a + 1$.

We shall examine the various errors introduced in a typical subinterval $[\alpha, \beta]$. The substitution of $\tilde{H}_0$, $\tilde{D}_0$, $\tilde{f}_0$ for H_0, D_0, f_0 in (8), (9), (10), (12) changes the error terms σ, τ_α, τ_β, τ to, say,

$$\tilde{\sigma} = \sigma + \sigma^I, \qquad \tilde{\tau}_\alpha = \tau_\alpha + \tau_\alpha^I,$$
$$\tilde{\tau}_\beta = \tau_\beta + \tau_\beta^I, \qquad \tilde{\tau} = \tau + \tau^I$$

where the terms σ^I, τ_α^I, τ_β^I, and τ^I are the interpolation errors resulting from replacing $u(x)$ and $u_x(x)$ in H, D, and f by $U(x)$ and $U_x(x)$. (See, for example, (23) and (24).) Quadrature/lumping errors are considered in § A.2 and interpolation errors in § A.1.

Of course, it is the effect of the local truncation errors on the global error that is important, especially since there is considerable latitude in how the local errors might be defined. Hence, we define the propagated local error to be the global error that would result from the use of a single finite element for $[\alpha, \beta]$ and the use of infinitesimal elements outside of this subinterval. This can be rather complicated, and so we model the problem outside of $[\alpha, \beta]$ by the differential equation $x^{-m}(x^m u_x/H(x))_x = Q(x)$ with linearized boundary conditions. Thus, the effect of the singularity is included. A lengthy argument indicates that the boundary conditions which give the largest errors are the following:

(i) *Singular case.* $v(a) = \text{given}$, $u(b) = \text{given}$, for which the propagated local error $\tilde{\tau}_s$ can be as large as

$$\tilde{\tau}_s = \int_\beta^b \frac{H(y)}{y^m}\, dy \cdot \tilde{\tau} + \tilde{H}_0 \tilde{\tau}_\alpha + \tilde{\sigma}; \tag{21}$$

and

(ii) *Regular case.* $u(a) = \text{given}$, $v(b) = \text{given}$ (which supposes that $a > 0$ or $m = 0$) for which the propagated local error $\tilde{\tau}_r$ can be as large as

$$\tilde{\tau}_r = \int_a^\alpha \frac{H(y)}{y^m}\, dy \cdot \tilde{\tau} + \tilde{H}_0 \tilde{\tau}_\beta - \tilde{\sigma}. \tag{22}$$

Because the primary purpose of our error analysis is to support our choice of interpolant and quadrature point, we stop just short of giving an explicit bound for either (21) or (22) and instead produce just the necessary constituents for such a bound.

A.1. Interpolation error. The local truncation errors due to interpolation are given by

$$\sigma^I := (H_0 - \tilde{H}_0) \int_\alpha^\beta v(x)\, dx \tag{23}$$

and

$$(24)\qquad \tau^I := (D_0-\tilde{D}_0)\int_\alpha^\beta x^m(u_t(\alpha)\psi_\alpha(x)+u_t(\beta)\psi_\beta(x))\,dx-(f_0-\tilde{f}_0)\int_\alpha^\beta x^m\,dx$$

together with suitable equations for τ_α^I and τ_β^I. All these errors involve partial derivatives of H, D, and f times the differences $u(x)-U(x)$ and $u_x(x)-U_x(x)$.

In the proofs that follow let $H(x):=H(x,u(x))$, $D(x):=D(x,u(x),u_x(x))$, and $f(x):=f(x,u(x),u_x(x))$. Also define $Q(x):=D(x)u_t(x)-f(x)$.

THEOREM 1. *The ψ interpolant satisfies*

$$\|u-U\|\leqq h^2C(P)$$

where $\|\cdot\|$ denotes the maximum norm on $[\alpha,\beta]$.

Proof. Consider first the case $\alpha>0$ or $m=0$. Integrating by parts, we obtain

$$u(x)=u(\alpha)+\int_\alpha^x\frac{dy}{y^m}H(x)x^mv(x)-\int_\alpha^x\int_\alpha^y\frac{dz}{z^m}(H(y)y^mv(y))_y\,dy.$$

A second equation is obtained by replacing α by β. Multiplying the first equation by $\psi_\alpha(x)$ and the second by $\psi_\beta(x)$, we obtain

$$u(x)=U(x)-\psi_\alpha(x)\int_\alpha^x\int_\alpha^y\frac{dz}{z^m}(H(y)y^mv(y))_y\,dy-\psi_\beta(x)\int_x^\beta\int_y^\beta\frac{dz}{z^m}(H(y)y^mv(y))_y\,dy.$$

Because

$$(H(x)x^mv(x))_x=x^m(H_x(x)v(x)+H(x)Q(x))$$

we have

$$|u(x)-U(x)|\leqq\omega(x)(\|H_xv+HQ\|)$$

where

$$\omega(x):=\psi_\alpha(x)\int_\alpha^x\int_\alpha^y\frac{dz}{z^m}y^m\,dy+\psi_\beta(x)\int_x^\beta\int_y^\beta\frac{dz}{z^m}y^m\,dy.$$

Integration once by parts gives

$$\omega(x)=\frac{1}{2(m+1)}\left\{(\beta^2-x^2)\int_\alpha^x\frac{dy}{y^m}-(x^2-\alpha^2)\int_x^\beta\frac{dy}{y^m}\right\}\bigg/\int_\alpha^\beta\frac{dy}{y^m}$$

and a second time gives

$$\omega(x)=\frac14\left\{(\beta^2-x^2)\int_\alpha^x\frac{y^2-\alpha^2}{y^2}\frac{dy}{y^m}+(x^2-\alpha^2)\int_x^\beta\frac{\beta^2-y^2}{y^2}\frac{dy}{y^m}\right\}\bigg/\int_\alpha^\beta\frac{dy}{y^m}.$$

Clearly,

$$\omega(x)\leqq\frac{(\beta^2-x^2)(x^2-\alpha^2)}{4x^2},$$

which is maximized for $x^2=\alpha\beta$ to yield the stated bound. For the case $\alpha=0$ and $m\geqq1$, we have

$$u(x)=u(\beta)-\int_x^\beta\int_0^y(H(z)z^mv(z))_z\,dz\frac{dy}{y^m}.$$

When we note that $U(x)=u(\beta)$ and

$$|(H(z)z^m v(z))_z| \leqq z^m C(P)$$

the theorem easily follows. □

In an identical fashion we can show that

$$\|u_t - U_t\| \leqq h^2 C(P). \tag{25}$$

Less accuracy can be expected for the approximation $U_x(\xi)$.

LEMMA. *If $k \neq 0$, then*

$$\frac{\gamma_k - \gamma}{k} \leqq \frac{h^2}{4\gamma}.$$

Furthermore,

$$\zeta \geqq \gamma_{-m}.$$

Proof of first inequality. We have $\gamma_k = \gamma I'/I$ where

$$I' := \gamma^{-k-2} \int_\alpha^\beta x^{k+1}\, dx$$

and

$$I := \gamma^{-k-1} \int_\alpha^\beta x^k\, dx.$$

With a change of variables $x =: \gamma(1+s)$ and $\theta := h/(2\gamma)$, we get

$$\begin{aligned} I &= \int_{-\theta}^{\theta} (1+s)^k\, ds \\ &= \int_0^\theta \{(1+s)^k + (1-s)^k\}\, ds. \end{aligned} \tag{26}$$

For I' we get (26) with m reduced by 1. Subtraction gives

$$\begin{aligned} I' - I &= \int_0^\theta s\{(1+s)^k - (1-s)^k\}\, ds \\ &= k\int_0^\theta \frac{\theta^2 - s^2}{2}\{(1+s)^{k-1} + (1-s)^{k-1}\}\, ds. \end{aligned} \tag{27}$$

Therefore, using (26) and (27), we have

$$\begin{aligned} \frac{k}{4}\frac{h^2}{\gamma} + \gamma - \gamma_k &= k\theta^2\gamma + \gamma - \gamma I'/I \\ &= \frac{\gamma}{I}\{k\theta^2 I + I - I'\} \\ &= \frac{k\gamma}{2I}\left\{2\theta^2 \int_0^\theta \{(1+s)^k + (1-s)^k\}\, ds \right. \\ &\quad \left. - \int_0^\theta (\theta^2 - s^2)\{(1+s)^{k-1} + (1-s)^{k-1}\}\, ds\right\} \\ &= \frac{k\gamma}{2I}\int_0^\theta \{(\theta^2 + 2s\theta^2 + s^2)(1+s)^{k-1} + (\theta^2 - 2s\theta^2 + s^2)(1-s)^{k-1}\}\, ds. \end{aligned}$$

The integral is nonnegative because $-2s\theta^2 \geqq -2s\theta$. □

Proof of second inequality. The inequality $\gamma_{-m} \leqq \zeta$ is a consequence of

$$\int_\alpha^\beta f(x)g(x)\,dx \leqq \left(\int_\alpha^\beta |f(x)|^{m+1}\,dx\right)^{1/(m+1)} \left(\int_\alpha^\beta |g(x)|^{(m+1)/m}\,dx\right)^{m/(m+1)}$$

with $f(x) = x^{1/(m+1)}$ and $g(x) = x^{-m^2/(m+1)}$. □

THEOREM 2. *The ψ interpolant satisfies*

$$|u_x(\xi) - U_x(\xi)| \leqq \left(\frac{\zeta}{\xi}\right)^m \left(1 + \frac{m}{\gamma}\right) h^2 C(P).$$

Proof. Defining $w(x) = x^m u_x(x)$, we note that

$$w_x = x^m(H_x v + HQ),$$
$$w_{xx} = x^m(H_{xx}v + 2H_xQ + HQ_x) + mx^{m-1}HQ,$$

and

$$Q_x = D_x U_t + D(Hv)_t - f_x. \tag{28}$$

If $m \geqq 1$ and $\alpha \to 0$, the bound goes to $+\infty$, and therefore assume either $m = 0$ or $\alpha > 0$. Hence we consider

$$U_x(\xi) - u_x(\xi) = \int_\alpha^\beta (w(x) - w(\xi))\frac{dx}{x^m} \Big/ \left(\xi^m \int_\alpha^\beta \frac{dx}{x^m}\right).$$

The numerator

$$\int_\alpha^\beta (w(x) - w(\xi))\frac{dx}{x^m} = w_x(\xi)\int_\alpha^\beta (x - \xi)\frac{dx}{x^m} + \int_\alpha^\beta \int_\xi^x \int_\xi^y w_{zz}(z)\,dz\,dy\frac{dx}{x^m}.$$

The first term vanishes because $\xi = \gamma_{-m}$. Replacing w_{zz} by bounds and using the definition of ζ, we obtain

$$|U_x(\xi) - u_x(\xi)| \leqq \frac{\zeta^{m+1}}{h\gamma\xi^m} \int_\alpha^\beta \int_\xi^x \int_\xi^y (z^m + mz^{m-1})\,dz\,dy\frac{dx}{x^m}\,C(P)$$
$$= \left(\frac{\gamma_1\zeta^{m+1} - \xi^{m+2}}{(m+2)\xi^m} + \frac{\zeta^{m+1} - \xi^{m+1}}{\xi^m}\right)\frac{C(P)}{m+1}.$$

We have

$$\begin{aligned}\gamma_1\zeta^{m+1} - \xi^{m+2} &= (\gamma_1 - \zeta)\zeta^{m+1} + \zeta^{m+2} - \xi^{m+2}\\ &\leqq (\gamma_1 - \zeta)\zeta^{m+1} + (\zeta - \xi)(m+2)\zeta^{m+1}\\ &\leqq (\gamma_1 - \gamma)\zeta^{m+1} + (\gamma - \xi)(m+2)\zeta^{m+1}\end{aligned}$$

where we use the fact that $\xi < \zeta < \gamma$. Similarly,

$$\zeta^{m+1} - \xi^{m+1} \leqq (\gamma - \xi)(m+1)\zeta^m.$$

The theorem follows from the lemma and the fact that $\zeta/\gamma < 1$. □

In the case where a is small but positive, linear interpolation is not as good. For example if $H \equiv 1$, $Q \equiv 0$, $v(a) = 1$, and $\alpha = a$, then $u_x(x) = (a/x)^m$ and the error for *linear* interpolation of $u(\xi)$ is

$$-\frac{m}{8}\frac{h^2}{a} + O(h^3)$$

and the error for $u_x(\xi)$ is

$$-\frac{m(m-1)}{24}\frac{h^2}{a^2}+O(h^3)$$

using the fact that $\xi=\gamma-(m/12)(h^2/\gamma)+O(h^4)$. Thus, the error is worse by a factor of $1/a$ than it is for ψ interpolation.

THEOREM 3. *For the singular case the ϕ interpolant satisfies*

$$\|u-U\|\leqq\frac{h^2\gamma}{m+1}C(\bar{P})\quad \textit{if } \beta\leqq c$$

and

$$\|u-U\|\leqq h^2C(P)\quad \textit{in any case.}$$

Proof of first inequality. Setting $w(x)=u_x(x)/x$ and integrating by parts, we obtain

$$u(x)=u(\alpha)+\frac{x^2-\alpha^2}{2}w(x)-\int_\alpha^x\frac{y^2-\alpha^2}{2}w_y(y)\,dy.$$

A second equation is obtained by replacing α by β. Multiplication of the first equation by $\phi_\alpha(x)$ and the second by $\phi_\beta(x)$ yields

$$u(x)=U(x)-\phi_\alpha(x)\int_\alpha^x\frac{y^2-\alpha^2}{2}w_y(y)\,dy-\phi_\beta(x)\int_x^\beta\frac{\beta^2-y^2}{2}w_y(y)\,dy.$$

Note that

$$w_x=H_x\frac{v}{x}+H\left(\frac{v}{x}\right)_x,\qquad \frac{v(x)}{x}=\frac{1}{x^{m+1}}\int_0^x y^mQ(y)\,dy,$$

and

$$\left(\frac{v(x)}{x}\right)_x=\frac{Q(x)}{x}-\frac{m+1}{x^{m+2}}\int_0^x y^mQ(y)\,dy \tag{29}$$
$$=\frac{1}{x^{m+2}}\int_0^x y^{m+1}\bar{Q}_y(y)\,dy$$

where the overbar indicates a spatial derivative defined across meshpoints. Hence $|v(x)/x|\leqq\|Q\|^*/(m+1)$ and $|(v(x)/x)_x|\leqq\|\bar{Q}_x\|^*/(m+2)$ where $\|\cdot\|^*$ denotes the maximum norm on $[0,\beta]$. Combining these facts with (28), we obtain

$$|u(x)-U(x)|\leqq w(x)C(\bar{P})/(m+1)$$

where

$$w(x)=\phi_\alpha(x)\int_\alpha^x\frac{y^2-\alpha^2}{2}dy+\phi_\beta(x)\int_x^\beta\frac{\beta^2-y^2}{2}dy$$
$$=\frac{(\beta-x)(x-\alpha)(\alpha\beta+2\gamma x)}{6\gamma}<\frac{h^2}{8}\gamma.\qquad\square$$

Proof of second inequality. From (29) we have $|w_x(x)|\leqq(\|H_xv\|+2\|H\|\,\|Q\|^*)/x$ and so

$$|u(x)-U(x)|\leqq\tilde{w}(x)C(P)$$

where

$$\tilde{w}(x)=\phi_\alpha(x)\int_\alpha^x \frac{y^2-\alpha^2}{2}\frac{dy}{y}+\phi_\beta(x)\int_x^\beta \frac{\beta^2-y^2}{2}\frac{dy}{y}.$$

Using $1/y \leqq \beta/y^2$, we get

$$\tilde{w}(x)\leqq \beta(\beta-x)(x-\alpha)/(\beta+\alpha)\leqq \frac{h^2}{4}. \qquad \square$$

In an identical fashion it can be shown that (25) holds for the ϕ interpolant in the singular case.

THEOREM 4. *For the singular case the ϕ interpolant satisfies*

$$|u_x(\xi)-U_x(\xi)|\leqq \frac{h^2\gamma}{m+1}C(\bar{P}) \quad \textit{if } \beta \leqq c$$

and

$$|u_x(\xi)-U_x(\xi)|\leqq \frac{m+1}{\gamma}h^2C(P) \quad \textit{in any case.}$$

Proof of first inequality. With $w(x):=u_x(x)/x$ we have

$$\begin{aligned}U_x(\xi)-u_x(\xi)&=\frac{2\xi}{\beta^2-\alpha^2}\int_\alpha^\beta (w(x)-w(\xi))x\,dx\\&=\frac{\xi}{h\gamma}\int_\alpha^\beta\int_\xi^x w_y(y)\,dy\,x\,dx\\&=\frac{\xi}{h\gamma}\int_\alpha^\beta\int_\xi^x \left(w_x(\xi)+\int_\xi^y w_{zz}(z)\,dz\right)dy\,x\,dx.\end{aligned}$$

Using the results in the proof of Theorem 3, we have that $|w_x(x)|\leqq C(\bar{P})/(m+1)$ and

$$w_{xx}=H_{xx}\frac{v}{x}+2H_x\left(\frac{v}{x}\right)_x+H\left(\frac{v}{x}\right)_{xx},$$

$$\left(\frac{v(x)}{x}\right)_{xx}=\frac{1}{x^{m+3}}\int_0^x y^{m+2}\bar{Q}_{yy}(y)\,dy,$$

$$\bar{Q}_{xx}=\bar{D}_{xx}u_t+2\bar{D}_x(Hv)_t+D(\overline{Hv})_{xt}-\bar{f}_{xx},$$

$$(\overline{Hv})_{xt}=\left(\bar{H}_xv+HQ-mH\frac{v}{x}\right)_t,$$

$$\left(\frac{v}{x}\right)_t=\frac{1}{x^{m+1}}\int_0^x y^mQ_t(y)\,dy,$$

$$Q_t=D_tu_t+Du_{tt}-f_t,$$

whence $|w_{xx}(x)|\leqq C(\bar{P})/(m+1)$. Therefore,

$$\begin{aligned}|u_x(\xi)-U_x(\xi)|&\leqq\frac{\xi}{h\gamma}\left(\left|\int_\alpha^\beta\int_\xi^x dy\,x\,dx\right|+\int_\alpha^\beta\int_\xi^x\int_\xi^y dz\,dy\,x\,dx\right)\frac{C(\bar{P})}{m+1}\\&=\xi\left(|\xi-\gamma_1|+\frac{1}{h\gamma}\int_\alpha^\beta\frac{(x-\xi)^2}{2}x\,dx\right)\frac{C(\bar{P})}{m+1}\\&\leqq\frac{h^2\gamma}{24}\frac{C(\bar{P})}{m+1}\end{aligned}$$

where the last inequality follows because $\xi=\gamma_1$. $\square$

Proof of second inequality. Modifying the previous proof, we get

$$U_x(\xi)-u_x(\xi)=\frac{\xi}{h\gamma}\int_\alpha^\beta\int_\xi^x\left(\xi^2 w_x(\xi)+\int_\xi^y (z^2w_z)_z\,dz\right)\frac{dy}{y^2}\,x\,dx.$$

We have that

$$x^2w_x=x\left(H_xv+HQ-(m+1)H\frac{v}{x}\right)$$

and

$$(x^2w_x)_x=m\left((m+1)H\frac{v}{x}-HQ-2H_xv\right)+x(H_{xx}v+2H_xQ+HQ_x),$$

whence

$$|x^2w_x|\leqq xC(P)\quad\text{and}\quad|(x^2w_x)_x|\leqq(m+x)C(P).$$

Therefore,

$$|u_x(\xi)-U_x(\xi)|\leqq\frac{\xi}{h\gamma}\left(\left|\int_\alpha^\beta\int_\xi^x\frac{dy}{y^2}\,x\,dx\right|\xi+\int_\alpha^\beta\int_\xi^x\int_\xi^y(m+z)\,dz\frac{dy}{y^2}\,x\,dx\right)C(P).$$

When we use $\xi=\gamma_1$,

$$\text{first term}=h|\gamma-\xi|\leqq\frac{h^3}{12\gamma}.$$

The

$$\begin{aligned}\text{second term}&\leqq\int_\alpha^\beta\int_\xi^x\int_\xi^x(m+z)\,dz\frac{dy}{y^2}\,x\,dx\\&=\frac{1}{\xi}\int_\alpha^\beta\left(m+\frac{x+\xi}{2}\right)(x-\xi)^2\,dx\\&\leqq(m+\xi)\frac{h^3}{12\gamma}\end{aligned}$$

when we use $\xi=\gamma_1=\gamma+h^2/(12\gamma)$. Putting this together, we get

$$|u_x(\xi)-U_x(\xi)|\leqq\frac{h^3}{12\gamma}(m+1+\xi).\qquad\square$$

For the singular case linear interpolation is not as good. For example, if $H\equiv1$, $Q\equiv1$, and $\alpha=0$, then $u_x(x)=x/(m+1)$ and the error

$$\frac{u(h)-u(0)}{h}-u_x(\xi)=-\frac{mh}{2(m+1)(m+2)}.$$

A.2. Quadrature and lumping error. The local truncation errors due to quadrature and lumping are given by

$$\sigma = \int_\alpha^\beta (H(x) - H(\xi))v(x)\,dx,$$

(30)
$$\tau_\alpha = \int_\alpha^\beta \int_x^\beta \frac{dy}{y^m} x^m\{D(x)u_t(x) - f(x) - D(\xi)u_t(\alpha) + f(\xi)\}\,dx,$$
$$\tau_\beta = \int_\alpha^\beta \int_\alpha^x \frac{dy}{y^m} x^m\{D(x)u_t(x) - f(x) - D(\xi)u_t(\beta) + f(\xi)\}\,dx,$$

$$\tau = \int_\alpha^\beta x^m\{D(x)u_t(x) - f(x) - D(\xi)U_t(x) + f(\xi)\}\,dx.$$

Because there can be cancellation between τ and either τ_α or τ_β (see Theorem 8, Proof of first inequality), we obtain bounds on

(31)
$$\tau_R := \int_a^\alpha \frac{H(y)}{y^m}\,dy \cdot \tau + H_0\tau_\beta$$

in the regular case and on

(32)
$$\tau_S := \int_\beta^b \frac{H(y)}{y^m}\,dy \cdot \tau + H_0\tau_\alpha$$

in the singular case. These come from the worst case propagated local errors given by (21) and (22).

THEOREM 5. *With $\xi = \gamma_{-m}$ the truncation error* (8) *satisfies*

$$|\sigma| \leqq h^3 C(P).$$

Proof. From (30) we have

(33)
$$\sigma = \int_\alpha^\beta \int_\xi^x H_y(y)\,dy\,v(x)\,dx$$
$$= \int_\alpha^\beta \int_\xi^x \left(H_x(\xi) + \int_\xi^y H_{zz}(z)\,dz\right) dy\,v(x)\,dx.$$

Using $v(x) = x^{-m}(\alpha^m v(\alpha) + \int_\alpha^x y^m Q(y)\,dy)$ and integrating by parts, we obtain

(34)
$$\sigma = \int_\alpha^\beta (x-\xi)\frac{dx}{x^m} H_x(\xi)\alpha^m v(\alpha) + \int_\alpha^\beta \int_x^\beta (y-\xi)\frac{dy}{y^m} x^m H_x(\xi) Q(x)\,dx$$
$$+ \int_\alpha^\beta \int_\xi^x \int_\xi^y H_{zz}(z)\,dz\,dy\,v(x)\,dx.$$

Because $\xi = \gamma_{-m}$, the first term vanishes and the inner integral of the second term is positive. Therefore,

$$|\sigma| \leqq \int_\alpha^\beta \left\{\int_x^\beta (y-\xi)\frac{dy}{y^m} x^m + \int_\xi^x \int_\xi^y dz\,dy\right\} dx \cdot C(P)$$
$$= \int_\alpha^\beta \frac{x-\xi}{x^m} \frac{x^{m+1} - \alpha^{m+1}}{m+1}\,dx \cdot C(P) + \frac{1}{2}\int_\alpha^\beta (x-\xi)^2\,dx \cdot C(P).$$

Because $\xi = \gamma_{-m}$,

$$\text{first term} = \frac{1}{m+1}\int_\alpha^\beta (x-\xi)x\,dx\,C(P)$$
$$= \frac{H\gamma}{m+1}(\gamma_1 - \xi)C(P)$$
$$\leqq \frac{h^3}{4}C(P)$$

when we use the lemma. The second term is bounded by $(h^3/6)C(P)$. □

In the case where a is small but positive, a choice of ξ other than γ_{-m} is not likely to be as good. For example, if $H(x) = 1 + x$, $Q \equiv 0$, $v(a) = 1$, and $\alpha = a$, then the error

$$\sigma = (\gamma_{-m} - \xi)a^m \int_a^\beta \frac{dx}{x^m};$$

and if, for example, $\xi = \gamma$, then

$$\sigma = -\frac{m}{12}\frac{h^3}{a} + O(h^4).$$

This is worse by a factor of $1/a$ than the bound given by Theorem 5.

THEOREM 6. *For the singular case with* $\xi = \gamma_1$

$$|\sigma| \leqq \frac{h^3\gamma}{m+1}C(\bar{P}) \quad \textit{if } \beta \leqq c$$

and

$$|\sigma| \leqq h^3 C(P) \quad \textit{in any case.}$$

Proof of first inequality. Using the fact that $\xi = \gamma_1$, we have from (33) that

$$\sigma = \int_\alpha^\beta (x-\xi)x\int_\xi^x \left(\frac{v}{y}\right)_y dy\,dx \cdot H_x(\xi) + \int_\alpha^\beta x\int_\xi^x\int_\xi^y H_{zz}(z)\,dz\,dy\frac{v(x)}{x}\,dx.$$

Using bounds from the proof of Theorem 3, we obtain

$$|\sigma| \leqq \frac{3}{2}\int_\alpha^\beta x(x-\xi)^2\,dx\frac{C(\bar{P})}{m+1} \leqq \frac{h^3}{8}\gamma\frac{C(\bar{P})}{m+1}$$

where the last inequality follows because $\xi = \gamma_1$. □

Proof of second inequality. Equation (34) gives

$$|\sigma| \leqq \left(\left|\int_\alpha^\beta \frac{x-\xi}{x^m}dx\right|\frac{\alpha^{m+1}}{m+1} + \int_\alpha^\beta \left|\int_x^\beta \frac{y-\xi}{y^m}dy\right| x^m\,dx\right)\|H_x\|\,\|Q\|$$
$$+ \int_\alpha^\beta \frac{(x-\xi)^2}{2}dx\,\|H_{xx}\|\,\|v\|.$$

Let $\eta < \beta$ be such that

$$\int_\eta^\beta \frac{x-\xi}{x^m}dx = 0.$$

Then

$$|\sigma| \leqq \left(-\int_\alpha^\eta \frac{x-\xi}{x^m} dx \frac{\alpha^{m+1}}{m+1} - \int_\alpha^\eta \int_x^\eta \frac{y-\xi}{y^m} dy\, x^m\, dx + \int_\eta^\beta \int_x^\beta \frac{y-\xi}{y^m} dy\, x^m\, dx \right) C(P)$$
$$+\frac{h^3}{6} C(P).$$

Interchanging the order of integration in the two double integrals and using twice again the definition of η, we obtain

$$|\sigma| \leqq 2 \int_\eta^\beta \frac{x-\xi}{x^m} \frac{x^{m+1}-\eta^{m+1}}{m+1} dx \cdot C(P) + \frac{h^3}{6} C(P)$$
$$\leqq 2 \int_\eta^\beta (x-\xi)(x-\eta)\, dx \cdot C(P) + \frac{h^3}{6} C(P).$$

The

$$\text{integral} \leqq \int_\xi^\beta (x-\xi)(x-\eta)\, dx$$
$$= \frac{1}{3}(\beta-\xi)^3 + \frac{1}{2}(\beta-\xi)^2(\xi-\eta) \leqq \frac{5}{6}(\beta-\xi)^3 \leqq \frac{5}{48} h^3. \qquad \square$$

A choice of ξ other than γ_1 is not likely to be as good. If $H(x) = 1+x$ and $Q \equiv 1$, then the error

$$\sigma = \frac{\gamma_1 - \xi}{m+1} h\gamma;$$

and if, for example, $\xi = \gamma$, then

$$\sigma = \frac{h^3}{12(m+1)}.$$

This is worse by a factor of $1/\gamma$ than the bound given by the first inequality of Theorem 6.

THEOREM 7. *For the regular case with $\xi = \gamma_{-m}$ the propagated truncation error τ_R defined by equation* (31) *satisfies*

$$|\tau_R| \leqq \left(1 + m\beta^{m-1} \int_a^\beta \frac{dy}{y^m}\right) h^3 C(P).$$

Proof. We can write $\tau = \tau_1 + \tau_2 + \tau_3$ where

$$\tau_1 = \int_\alpha^\beta x^m (D(x) - D(\xi))(u_t(x) - u_t(\xi))\, dx,$$

$$\tau_2 = \int_\alpha^\beta x^m \{(D(x) - D(\xi)) u_t(\xi) - (f(x) - f(\xi))\}\, dx,$$

$$\tau_3 = D(\xi) \int_\alpha^\beta x^m (u_t(x) - U_t(x))\, dx.$$

Clearly,

$$|\tau_1| \leqq \int_\alpha^\beta x^m (x-\xi)^2\, dx\, \|D_x\|\, \|u_{xt}\|.$$

Expanding D and f in τ_2 in a Taylor series about $x=\xi$, we obtain $\tau_2=\tau_2'+\tau_2''$ where

$$\tau_2'=\int_\alpha^\beta x^m(x-\xi)\,dx\,(D_x(\xi)u_t(\xi)-f_x(\xi))$$

and

$$\tau_2''=\int_\alpha^\beta x^m\frac{(x-\xi)^2}{2}(D_{xx}(\xi')u_t(\xi)-f_{xx}(\xi'))\,dx.$$

Applying the lemma, we have

$$|\tau_2'|\leqq\int_\alpha^\beta x^m\,dx(\gamma_m-\gamma_{-m})C(P)$$
$$\leqq\frac{m}{2}h^3\beta^{m-1}C(P).$$

Also we have

$$|\tau_2''|\leqq\frac{1}{2}\int_\alpha^\beta x^m(x-\xi)^2\,dx\,(\|D_{xx}\|\,|u_t(\xi)|+\|f_{xx}\|)$$

and

$$|\tau_3|\leqq|D(\xi)|\int_\alpha^\beta x^m\,dx\,\|u_t-U_t\|,$$

and so

$$|\tau|\leqq\left(1+\frac{m}{\beta}\right)h^3\beta^mC(P).$$

The other part of τ_R can be expressed

$$\tau_\beta=\int_\alpha^\beta\int_\alpha^x\frac{dy}{y^m}x^m\{Q(x)-Q(\xi)+D(\xi)(u_t(\xi)-u_t(\beta))\}\,dx,$$

and so

$$|\tau_\beta|\leqq h^2\beta^m\int_\alpha^\beta\frac{dx}{x^m}C(P).$$

Note that for $x\leqq\beta$

$$\beta^m=x^m+\beta^m-x^m\leqq x^m+m(\beta-x)\beta^{m-1}\leqq x^m+mh\beta^{m-1},$$

and so

$$\beta^m\int_\alpha^\beta\frac{dx}{x^m}\leqq h+mh\beta^{m-1}\int_\alpha^\beta\frac{dx}{x^m}.$$

Therefore,

$$|\tau_R|\leqq h^3\left(\beta^m\int_a^\alpha\frac{dx}{x^m}+m\beta^{m-1}\int_a^\beta\frac{dx}{x^m}+1\right)C(P).$$

However,

$$\beta^m\int_a^\alpha\frac{dx}{x^m}\leqq\alpha-a+m(\beta-a)\beta^{m-1}\int_a^\alpha\frac{dx}{x^m}\leqq1+m\beta^{m-1}\int_a^\beta\frac{dx}{x^m},$$

from which the result follows. □

THEOREM 8. *For the singular case with $\xi=\gamma_1$ the propagated truncation error τ_S defined by equation* (32) *satisfies*

$$|\tau_S| \leqq \left(\beta \log\frac{c}{\beta}+\frac{\beta}{c}\right) h^3 C(\bar{P}) \quad \text{if } \beta \leqq c \text{ and } m \geqq 2$$

and

$$|\tau_S| \leqq h^3 C(P) \quad \text{in any case.}$$

Proof of first inequality. We must modify the proof of Theorem 7. We have

$$\tau = \tau_2' + (\tau_1 + \tau_2'' + \tau_3)$$

where

$$|\tau_1+\tau_2''+\tau_3| \leqq h^2 \int_\alpha^\beta x^m \, dx \, C(P). \tag{35}$$

Also

$$\tau_\alpha = \int_\alpha^\beta \int_x^\beta \frac{dy}{y^m} x^m (x-\xi)\, dx \,(D_x(\xi)u_t(\xi) - f_x(\xi)) + \tau_\alpha'$$

where

$$\tau_\alpha' = \int_\alpha^\beta \int_x^\beta \frac{dy}{y^m} x^m \Big\{ (D(x)-D(\xi))(u_t(x)-u_t(\xi)) + \frac{(x-\xi)^2}{2}(D_{xx}(\xi')u_t(\xi) - f_{xx}(\xi')) + D(\xi)(u_t(x)-u_t(\alpha)) \Big\}\, dx.$$

Use of the fact that $|u_{tx}(x)| \leqq xC(P)/(m+1)$ gives us

$$\begin{aligned} |\tau_\alpha'| &\leqq \int_\alpha^\beta \int_x^\beta \frac{dy}{y^m} x^m \left\{ h^2 + \frac{1}{m+1}\int_\alpha^x y\, dy \right\} dx \cdot C(P) \\ &\leqq \left\{ \int_\alpha^\beta \frac{x^{m+1}-\alpha^{m+1}}{(m+1)x^m}\, dx + \frac{1}{m+1}\int_\alpha^\beta y\, dy \right\} h^2 C(P) \\ &\leqq h^3 \gamma C(P)/(m+1). \end{aligned}$$

We have

$$\int_\beta^1 \frac{H(y)}{y^m}\, dy = \int_\beta^c \frac{H_0}{y^m}\, dy + \int_\beta^c \frac{H(y)-H_0}{y^m}\, dy + \int_c^1 \frac{H(y)}{y^m}\, dy$$

where the

$$|\text{last two terms}| \leqq \int_\beta^c \frac{dy}{y^{m-1}} \|\bar{H}_x\|^* + \int_c^1 \frac{dy}{y^m} \|H\|^*.$$

Combining all this, we obtain

$$\begin{aligned} |\tau_S| \leqq &\left| \int_\alpha^\beta \int_x^c \frac{dy}{y^m} x^m (x-\xi)\, dx\, H_0 (D_x(\xi)u_t(\xi) - f_x(\xi)) \right| \\ &+ \left(\int_\beta^c \frac{dy}{y^{m-1}} \|\bar{H}_x\|^* + \int_c^1 \frac{dy}{y^m}\|H\|^* \right) |\tau_2'| \\ &+ \int_\beta^1 \frac{H(y)}{y^m}\, dy\, h^2 \int_\alpha^\beta x^m\, dx\, C(P) + H_0 h^3 \gamma C(P)/(m+1). \end{aligned}$$

Integrating the inner integral and using the definition of ξ, we have

$$\left|\int_\alpha^\beta \int_x^c \frac{dy}{y^m} x^m(x-\xi)\,dx\right| = \left|\frac{-c^{1-m}}{m-1}\int_\alpha^\beta x^m(x-\xi)\,dx\right|.$$

Furthermore,

$$\begin{aligned}\int_\alpha^\beta x^m(x-\xi)\,dx &= \int_\alpha^\beta x(x^{m-1}-\xi^{m-1})(x-\xi)\,dx\\ &\leqq \int_\alpha^\beta x(m-1)\beta^{m-2}(x-\xi)^2\,dx\\ &\leqq (m-1)\beta^{m-1}\frac{h^3}{3}.\end{aligned}$$

Also, we have

$$|\tau_2'| \leqq \int_\alpha^\beta x^m(x-\xi)\,dx\,C(P) \leqq (m-1)\beta^{m-1}h^3C(P). \tag{36}$$

Putting all this together, we get

$$|\tau_S| \leqq \left\{\left(\frac{\beta}{c}\right)^{m-1} + (m-1)\beta^{m-1}\int_\beta^c \frac{dy}{y^{m-1}} + (m-1)\beta^{m-1}\int_c^1 \frac{dy}{y^m} + \beta^m\int_\beta^1 \frac{dy}{y^m} + \gamma\right\} h^3C(\bar{P}).$$

The result follows from

$$\begin{aligned}(m-1)\beta^{m-1}\int_\beta^c \frac{dy}{y^{m-1}} &\leqq \beta\int_\beta^c \frac{dy}{y} + (m-2)\beta^{m-1}\int_\beta^c \frac{dy}{y^{m-1}}\\ &\leqq \beta\log\frac{c}{\beta} + \beta\end{aligned}$$

and

$$\int_c^1 \frac{dy}{y^m} \leqq \frac{1}{(m-1)c^{m-1}} \quad \text{and} \quad \int_\beta^1 \frac{dy}{y^m} \leqq \frac{1}{(m-1)\beta^{m-1}}.$$

Proof of second inequality. We modify the proof of the first inequality. We have from (35) and (36) that

$$|\tau| \leqq h^3(\beta^m + (m-1)\beta^{m-1})C(P).$$

Also,

$$\int_\beta^1 \frac{H(y)}{y^m}\,dy \leqq \frac{C(P)}{(m-1)\beta^{m-1}}.$$

To conclude the proof we split τ_α as

$$\tau_\alpha = \int_\alpha^\beta \int_x^\beta \frac{dy}{y^m} x^m\{Q(x) - Q(\xi) + D(\xi)(u_t(\xi) - u_t(\alpha))\}\,dx,$$

and so

$$|\tau_\alpha| \leqq \int_\alpha^\beta \int_x^\beta dy\,dx\,h(\|Q_x\| + |D(\xi)|\,\|u_{xt}\|). \qquad \square$$

Remark. For $x \leqq c$, we can obtain results for $1 < m < 2$ and for $m > 2$ that are better than those given by Theorem 8.

A choice of ξ other than γ_1 is not likely to be as good. If $H(x) \equiv 1$, $Q(x) = x$, and $\alpha = 0$, then the error

$$\tau_S = \int_h^1 \frac{dy}{y^m} \int_0^h x^m(x-\xi)\,dx + \int_0^h \int_x^h \frac{dy}{y^m} x^m(x-\xi)\,dx$$
$$= \begin{cases} \frac{1}{2}(\gamma_1-\xi)h^2 \log h + O(h^3), & m=1, \\ \dfrac{(\gamma_1-\xi)h^2}{2(m-1)} + O(h^4), & m>1. \end{cases}$$

If, for example, $\xi = \gamma$, this is worse by a factor $\log(1/h)$ for $m=1$ and by a factor of $1/(h \log(1/h))$ for $m=2$ than the bounds given by Theorem 8.

Acknowledgment. Thanks are due to Peter Dew for suggesting this joint work.

REFERENCES

[1] M. BAKKER, *Software for the semi-discretization of time-dependent partial differential equations in one space variable*, Report NW 52/77, Department of Numerical Mathematics, Mathematisch Centrum, Amsterdam, the Netherlands, 1977.

[2] M. BERZINS AND P. M. DEW, *A note on the testing of semi-discrete methods for the solution of second order non-linear parabolic* P.D.E.*s in one space variable*, Report 128, Department of Computer Studies, The University, Leeds, UK, 1980.

[3] ———, *A note on C^0 Chebyshev methods for parabolic* P.D.E.*s*, IMA J. Numer. Anal., 7 (1987), pp. 15-37.

[4] M. BERZINS, P. M. DEW, AND R. M. FURZELAND, *Developing* P.D.E. *software using the method of lines and differential-algebraic integrators*, Appl. Numer. Math. (1989), to appear.

[5] M. BERZINS AND R. M. FURZELAND, *A user's manual for* SPRINT—*a versatile software package for solving systems of algebraic, ordinary and partial differential equations*, TNER 85.058, Shell Research Limited, Thornton Research Centre, Chester, UK, 1985.

[6] ———, *A user's manual for* SPRINT: *Part* 2 *solving partial differential equations*, Report 202, Department of Computer Studies, The University, Leeds, UK, 1985.

[7] M. M. CHAWLA AND C. P. KATTI, *A finite-difference method for a class of singular two-point boundary-value problems*, IMA J. Numer. Anal., 4 (1984), pp. 457-466.

[8] P. M. DEW AND J. E. WALSH, *A set of library routines for the numerical solution of parabolic equations in one space variable*, ACM Trans. Math. Software, 7 (1981), pp. 295-314.

[9] K. ERIKSSON AND V. THOMÉE, *Galerkin methods for singular boundary value problems in one space dimension*, Math. Comp., 42 (1984), pp. 345-367.

[10] D. JESPERSEN, *Ritz-Galerkin methods for singular boundary value problems*, SIAM J. Numer. Anal., 15 (1978), pp. 813-834.

[11] J. KAUTSKY AND N. K. NICHOLS, *Equidistributing meshes with constraints*, SIAM J. Sci. Statist. Comput., 1 (1980), pp. 499-511.

[12] L. PETZOLD, *Differential/algebraic equations are not* ODE's, SIAM J. Sci. Statist. Comput., 3 (1982), pp. 367-384.

[13] ———, *Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 136-148.

[14] R. D. RUSSELL AND L. F. SHAMPINE, *Numerical methods for singular boundary value problems*, SIAM J. Numer. Anal., 12 (1975), pp. 13-36.

[15] N. L. SCHRYER, *Numerical solution of coupled systems of partial differential equations in one space variable and time*, in Elliptic Problem Solvers, M. Schultz, ed., Academic Press, London, 1981, pp. 413-417.

[16] R. F. SINCOVEC AND N. K. MADSEN, *Software for nonlinear partial differential equations*, ACM Trans. Math. Software, 1 (1975), pp. 232-260.

[17] R. SKEEL, *Improving routines for parabolic equations in one space dimension*, Numerical Analysis Report 63, Department of Mathematics, The University, Manchester, UK, 1981.

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

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 1, pp. 33–50, January 1990

MULTIGRID FOR THE ONE-DIMENSIONAL INVISCID BURGERS EQUATION*

WIM A. MULDER†

Abstract. A multigrid method for computing steady inviscid compressible flow is investigated for the one-dimensional scalar case. The discretisation in space is obtained by upwind differencing and has first- or second-order accuracy. Only relaxation schemes that affect the solution locally are examined. To obtain some insight into the convergence behaviour, two-level convergence analysis is carried out for the linear constant-coefficient case. The resulting two-grid convergence rates are compared to the asymptotic convergence rates observed in numerical experiments on the nonlinear one-dimensional inviscid Burgers equation.

For a test problem with a smooth steady solution, the observed asymptotic convergence rates agreed within $O(h)$ with the linear two-grid convergence rates. A discontinuous solution displayed slower convergence, due to the shock and the sonic point. Although the resulting convergence rate was still acceptable, it could be improved through local relaxation and regularisation of the shock. In this way, a first-order-accurate solution could be obtained in one F-cycle per grid, using damped Point-Jacobi relaxation and successive grid refinement. Second-order accuracy required about eight cycles, using the Defect Correction technique.

Key words. multigrid method, hyperbolic conservation laws, shock waves

AMS(MOS) subject classifications.35L67, 65B99, 76A60

1. Introduction. The multigrid technique is a powerful tool for the construction of $O(N)$ methods for a wide class of problems. Its application to the solution of elliptic partial differential equations has received much attention, and the technique is well established for these problems (cf. [3], [6], [20], and references therein). It took some time before any results for the computation of steady solutions to hyperbolic equations, specifically the Euler equations of gas dynamics, were obtained, but successful experiments are now available [1], [7]–[11], [13], [14], [16]. Theoretical estimates of multigrid convergence rates for hyperbolic equations are scarce and incomplete.

In this paper, the convergence towards the steady state by means of the multigrid method proposed in [13] is investigated for the one-dimensional scalar case. Steady solutions to a simple nonlinear one-dimensional hyperbolic equation, the inviscid Burgers equation, are considered. Of course, it does not make much sense to apply the multigrid technique to one-dimensional problems, because direct $O(N)$ methods are available (cf. [12]). Nevertheless, linear two-level analysis is carried out to estimate the two-grid convergence factor. The results are compared with the numerical experiments on the nonlinear equation. Special attention is paid to the effect of the sonic point and the shock on the convergence rate. The full system of (linearised) Euler equations in two dimensions is considered elsewhere [15].

Two periodic test problems, one with a continuous and one with a discontinuous steady solution, are described in §2.1. The spatial discretisation is obtained by upwind differencing, using the schemes of Godunov, Enquist-Osher, or Roe (cf. [23]). For the

*Received by the editors August 12, 1986; accepted for publication (in revised form) October 21, 1987. This work was supported by the Center for Large Scale Scientific Computing (CLaSSiC) Project at Stanford University, under Office of Naval Research contract N00014-82-K-0335.

†Department of Computer Science, Stanford University, Stanford, California 94305-2140. Present address, Koninklijke/Shell Exploratie en Produktie Laboratorium, Postbus 60, 2280 AB Rijswijk, the Netherlands.

latter, the version that obeys the entropy condition is adopted (§2.2). The discretisation has first-order accuracy. The construction of second-order-accurate schemes is reviewed in §2.3. The singularity at the shock is examined in §2.4.

Two-grid convergence factors are estimated by two-level local-mode analysis [3] in §3. We only consider relaxation schemes that affect the solution locally, because the multigrid method should handle the global aspects of the solution. The analysis is carried out for a linear hyperbolic equation with a constant coefficient. We may expect the result to approximate the two-grid convergence rate in the variable coefficient and nonlinear case, as long as the convection speed varies gradually with x and does not change sign. This simplification certainly cannot explain the convergence behaviour at the shock and sonic point. The convergence rate at the shock is considered in §3.5, and at the sonic point in the Appendix.

Numerical experiments are described in §4. The multigrid method is implemented as a Full Approximation Storage (FAS) scheme, including successive grid refinement. This combination is also known as Full Multigrid (FMG). A comparison is made between predicted and observed convergence factors. Only damped Point-Jacobi (PJ) and one type of Red-Black relaxation (RB) are used, since these schemes emerge as the best in the previous section.

The main conclusions are summarised in §5.

2. Spatial discretisation.

2.1. The problem. We consider two types of steady solutions to the one-dimensional scalar hyperbolic equation with periodic boundary conditions

$$u_t + (\tfrac{1}{2}u^2)_x = s(x), \tag{2.1}$$

the first being smooth, the second discontinuous. The solutions are periodic in x on the interval $[0,1)$. Setting

$$u(x) = c_0 + \sin 2\pi(x-\xi), \qquad c_0 = 2, \tag{2.2a}$$

we are led to a source term

$$s(x) = 2\pi\ (c_0 + \sin 2\pi(x-\xi))\cos 2\pi(x-\xi). \tag{2.2b}$$

Although the steady state corresponding to this source term is continuous, a shock may occur during the evolution toward the steady state.

A discontinuous steady solution is found for

$$s(x) = \tfrac{1}{2}\pi\ \sin 2\pi(x-\xi), \tag{2.3a}$$

namely,

$$u(x) = \begin{cases} \sin\pi(x-\xi) & \text{for } 0 \le x < \xi + \frac{1}{2}, \\ -\sin\pi(x-\xi) & \text{for } \xi + \frac{1}{2} < x \le 1. \end{cases} \tag{2.3b}$$

For $0 < \xi < \frac{1}{2}$, this solution has a sonic point at $x = \xi$ and a shock at $x = \xi + \frac{1}{2}$ (cf. [23]). Numerical examples will be given for $\xi = 0.1$. It should be remarked that (2.1) must be interpreted as the limit for zero viscosity ϵ of the same equation with ϵu_{xx} added to the right-hand side. This is equivalent to imposing an entropy condition in addition to (2.1) and provides uniqueness of the solution. The average value of the initial data is conserved and must, therefore, be the same for the initial data and the steady state.

2.2. Upwind differencing. The spatial discretisation of (2.1) is obtained by upwind differencing, using Godunov's [5], the Enquist–Osher (E–O) [4], [17], or Roe's scheme [18], [23]. The E–O scheme is identical to Flux-Vector Splitting (FVS) [19] for the one-dimensional scalar case.

A computational grid with N zones is defined by $x_k = (k+\frac{1}{2})h$, where the cell size $h = 1/N$ and $k = 0, \cdots, N-1$. Equation (2.1) is discretised in space by averaging per volume:

$$u_k^h = (I^h u)_k \equiv \frac{1}{h}\int_{x_k-h/2}^{x_k+h/2} u(x)\, dx. \tag{2.4}$$

The same discretisation operator I^h is applied to the residual

$$r(u,x) = s(x) - (\tfrac{1}{2}u^2)_x, \tag{2.5}$$

yielding the discrete first-order approximation

$$r_k = s_k - \frac{1}{h}(f_{k+1/2} - f_{k-1/2}). \tag{2.6}$$

The fluxes $f_{k\pm1/2}$ are evaluated by upwind differencing. Godunov's scheme lets

$$f_{k+1/2} = f(u_k, u_{k+1}) = \tfrac{1}{2}\max\left[\max(0,u_k)^2, \min(0,u_{k+1})^2\right]. \tag{2.7}$$

The E–O scheme is equivalent to FVS when applied to Burgers' equation. For the latter we have

$$f(u_k, u_{k+1}) = f^+(u_k) + f^-(u_{k+1}), \tag{2.8a}$$

where

$$f^+(u) + f^-(u) = f(u), \qquad f^+(u) = \tfrac{1}{2}[\max(0,u)]^2. \tag{2.8b}$$

The E–O, or FVS scheme, differs from Godunov's only at the shock. Roe's scheme is identical to Godunov's, except at the sonic point, where the modification that suppresses expansion shocks lets

$$f_{k+1/2} = f(u_k, u_{k+1}) = \tfrac{1}{2}u_k u_{k+1} \quad \text{if } u_k \le 0 \le u_{k+1}. \tag{2.9}$$

With this flux, the solution at the sonic point is smooth, whereas Godunov's scheme sets $f_{k+1/2} = 0$, causing an $O(h)$ jump. However, if the stationary sonic point is positioned precisely at the cell interface, Roe's scheme (2.9) allows for two solutions: a smooth one and a discontinuous one. This problem will be ignored here by avoiding such a situation.

The numerical steady solution $\overline{u}_k^h$ obeys $r_k(\overline{u}_{k-1}, \overline{u}_k, \overline{u}_{k+1}) = 0$ for $i = 0, \cdots, N-1$. It is first-order accurate, i.e., $\|\overline{u}_k^h - u_{e\,k}^h\| = O(h)$, where $u_{e\,k}^h \equiv I^h u(x)$ is the average per cell of the exact solution. The norm is the ℓ_1 norm. Godunov's and Roe's scheme are first-order accurate in the ℓ_∞ norm as well. The E–O, or FVS scheme, smears out the shock over two cells, thus causing a local $O(1)$ error.

2.3. Second-order accuracy. Second-order spatial accuracy can be obtained by assuming the solution to be piecewise linear rather than piecewise constant [22]. The idea is to write the solution as

$$u(x) = u_k + \left(\frac{x - x_k}{h}\right)\Delta_k, \qquad \frac{|x - x_k|}{h} < \frac{1}{2}. \tag{2.10a}$$

Here Δ_k/h is a discrete approximation to $I^h \partial u/\partial x$, with

$$\Delta_k = \text{ave}(u_k - u_{k-1}, u_{k+1} - u_k). \tag{2.10b}$$

In smooth regions we want $\text{ave}(\Delta_-, \Delta_+) = \frac{1}{2}(\Delta_- + \Delta_+)$, whereas limiting to the smaller of Δ_- and Δ_+ is required near discontinuities to preserve monotonicity (i.e., to avoid local under- or overshoots). We use an averaging-limiting procedure from [21]:

$$\text{ave}(\Delta_-, \Delta_+) = \frac{(\Delta_+^2 + \epsilon_a^2)\Delta_- + (\Delta_-^2 + \epsilon_a^2)\Delta_+}{\Delta_-^2 + \Delta_+^2 + 2\epsilon_a^2}. \tag{2.11}$$

The bias ϵ_a prevents division by zero. Clipping of smooth extrema is avoided if ϵ_a approximately equals the average value of $|u_k - u_{k-1}|$ in smooth regions of the flow. For the numerical examples mentioned above we adopt $\epsilon_a = 4\,h$ for the smooth problem (2.2) and $\epsilon_a = 2\,h$ for the discontinuous problem (2.3). Once the Δ_k are computed, the fluxes at $k + \frac{1}{2}$ are evaluated by $f(u_k + \frac{1}{2}\Delta_k, u_{k+1} - \frac{1}{2}\Delta_{k+1})$, using one of the schemes mentioned in §2.2.

2.4. The singularity at the shock. The numerical problem of determining the steady state to (2.1) can be written as

$$r_k(u_{k-p}, \cdots, u_{k+p}) = 0 \quad \text{for } k = 0, \cdots, N-1. \tag{2.12}$$

Here $p = 1$ for a first-order scheme and $p = 2$ for a second-order scheme. For the example given in (2.3), (2.12) is ill posed, as the position of the shock depends on the initial data. To obtain a unique steady solution, we need the additional requirement that

$$\frac{1}{N}\sum_{k=0}^{N-1} u_k = C \quad \text{(a constant)}. \tag{2.13}$$

The singularity at the shock is appropriate, because the original differential equation is singular. In the latter, the singularity is regularised by augmenting the differential equation with the jump relation across the shock. This is implicit in the conservation form of the equation. Here, however, conservation in time is abandoned to obtain fast convergence, and (2.13) must be imposed to obtain a unique stationary solution.

To describe the singularity in more detail, we need to know the structure of stationary shocks [23, §5]. Assume that (2.12) is obtained by Godunov's or Roe's scheme, and that the source term is absent. Let $u_L > 0$ be the state left of the shock, and $u_R = -u_L < 0$ at the right. Between these two states there is a cell M containing the shock: $u_L \geq u_M \geq u_R$. The corresponding residual for the first-order scheme is

$$r_M = -(\tfrac{1}{2}u_R^2 - \tfrac{1}{2}u_L^2), \tag{2.14}$$

which is independent of u_M. Thus, the state u_M never enters the discrete equations (2.12) and can be chosen arbitrarily, as long as $u_L > u_M > u_R$. We thus have N equations in $N-1$ unknowns. Condition (2.13) is required to determine a unique value for u_M.

If the E–O, or FVS scheme, is used, a steady shock is represented by a sequence of four cells: u_L, u_M, u_N, and u_R, where $u_L \geq u_M \geq 0 \geq u_N \geq u_R$ and $u_L = -u_R$. The two residuals,

$$\begin{aligned} r_M &= -\left[(\tfrac{1}{2}u_M^2 + \tfrac{1}{2}u_N^2) - \tfrac{1}{2}u_L^2\right], \\ r_N &= -\left[\tfrac{1}{2}u_R^2 - (\tfrac{1}{2}u_M^2 + \tfrac{1}{2}u_N^2)\right], \end{aligned} \tag{2.15}$$

show that both u_M and u_N enter the discrete equations, but only as the combination $\frac{1}{2}u_M^2 + \frac{1}{2}u_N^2$, thus leaving the individual values of u_M and u_N undetermined. We again have N equations in $N-1$ unknowns, and (2.13) must be imposed as an extra condition to obtain uniqueness.

The second-order-accurate discretisation is singular as well. However, the structure of stationary shocks is more complicated and will not be considered here.

3. Convergence factors.

3.1. Preliminaries. Two-grid convergence factors for a number of relaxation schemes will be estimated by means of two-level analysis in Fourier space [6]. Here we consider the linear equation $u_t + au_x = 0$, where a is a positive constant. This is, of course, a major simplification. Let the iteration error $v^h \equiv \overline{u}^h - u^h$, where $\overline{u}^h$ is the steady state. The discrete linear operator is

$$L^h = \frac{a}{h}(I - T^{-1}). \tag{3.1}$$

Here I is the identity and the shift operator T acts according to $T^s v_k = v_{k+s}$. The discrete Fourier transform of v_k^h will be denoted by $\hat{v}_l^h$. The shift operator in Fourier space is

$$\hat{T}(\theta) = \exp(i\theta), \qquad \theta \equiv 2\pi l/N, \qquad l = -(\tfrac{1}{2}N - 1), \cdots, \tfrac{1}{2}N. \tag{3.2}$$

The high frequencies, which cannot be represented on the next coarser grid, lie in the range $\frac{1}{2}\pi \leq |\theta| \leq \pi$. The coarse-grid correction (CGC) operator, and also the RB relaxation operator, couples the frequencies θ and $\theta + \pi$. Therefore, define

$$\hat{V}_l^h \equiv \begin{pmatrix} \hat{v}_l \\ \hat{v}_{l+N/2} \end{pmatrix}. \tag{3.3}$$

A 2×2 matrix $\hat{Z}$, operating on $\hat{V}_l^h$, must have the property

$$\hat{z}_{21}(\theta) = \hat{z}_{12}(\theta + \pi), \qquad \hat{z}_{22}(\theta) = \hat{z}_{11}(\theta + \pi). \tag{3.4}$$

Note that $\hat{T}(\theta + \pi) = -\hat{T}(\theta)$. The residual operator $\hat{L}^h$ vanishes for the longest wave $(\theta = 0)$, which reflects the fact that the solution is determined up to a constant. This wave will be ignored.

3.2. Coarse-grid correction (CGC) operator. The CGC operator describes the result of (i) restriction of the fine-grid residual to the next coarser grid, (ii) exact solution of the coarse-grid equations, and (iii) prolongation of the CGC to the fine grid. We choose a restriction operator I_h^{2h} that averages the values in two neighbouring cells $(2k, 2k+1)$, where $k = 0, 1, \cdots, \frac{1}{2}N - 1$. The prolongation operator I_{2h}^h describes piecewise constant interpolation from the coarse to the fine grid. Both are first-order operators. The CGC operator,

$$K = I - I_{2h}^h (L^{2h})^{-1} I_h^{2h} L^h, \tag{3.5}$$

has a Fourier representation

(3.6a)
$$\hat{K} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \frac{1}{2} \begin{pmatrix} 1+\hat{T}^{-1} \\ 1-\hat{T}^{-1} \end{pmatrix} \left(\hat{L}^{2h}\right)^{-1} \frac{1}{2} \begin{pmatrix} 1+\hat{T} & 1-\hat{T} \end{pmatrix} \begin{pmatrix} \hat{L}^h(\theta) & 0 \\ 0 & \hat{L}^h(\theta+\pi) \end{pmatrix}.$$

$\hat{K}$ is understood to operate on $\hat{V}^h$. Using (3.1) and $\hat{L}^{2h} = (a/2h)(1-\hat{T}^{-2})$, we obtain

$$\hat{K} = \frac{1}{2} \begin{pmatrix} 1-\hat{T} & 1+\hat{T} \\ 1-\hat{T} & 1+\hat{T} \end{pmatrix}. \tag{3.6b}$$

This matrix obeys property (3.4). In physical space, the CGC operator lets

$$\left.\begin{aligned} \tilde{v}_{2k} &= v_{2k} - v_{2k+1} \\ \tilde{v}_{2k+1} &= 0 \end{aligned}\right\} \quad k = 0, 1, \cdots, \tfrac{1}{2}N - 1, \tag{3.6c}$$

where v denotes the iteration error before the CGC, and $\tilde{v}$ denotes the iteration error after.

3.3. Relaxation. A relaxation scheme provides an approximate solution $\tilde{u}^h$ to the nonlinear problem $r^h = 0$. Here we will discuss relaxation schemes by starting with a time-accurate integration of the equations and then making simplifications. The resulting relaxation schemes are not time-accurate.

An example is the explicit scheme

$$\left[\frac{1}{\Delta t}\right](\tilde{u}^h - u^h) = r^h, \tag{3.7}$$

or the implicit scheme

$$\left[\frac{1}{\Delta t} - \alpha \left(\frac{dr}{du}\right)^h\right](\tilde{u}^h - u^h) = r^h. \tag{3.8}$$

The residual r^h is computed from the solution u^h, and $\tilde{u}^h$ is the solution after applying the explicit or implicit scheme. Both are based on the time-dependent problem and provide a steady state by a time-accurate integration of the differential equation. The choice $\alpha = 1$ yields a "backward Euler" scheme. If the time-step Δt becomes large, the latter reduces to Newton's method. This property is exploited by the Switch Evolution/Relaxation (SER) scheme, proposed in [12] and [24], where $\Delta t \propto 1/\|r^h\|$. After some initial time-accurate searching while the residuals are large, this scheme automatically switches to Newton's method when the solution approaches the steady state. The finite $1/\Delta t$ term allows for the solution of the linear system (3.8) even if $(dr/du)^h$ is singular.

Note that the implicit scheme (3.8) is not conservative in time, except for $\alpha = 0$. For other values of α, we still have *global* conservation in time on a periodic grid. For a residual of the form (2.6), and a periodic grid with N points, we obtain, after summation of the N equations represented by (3.8),

$$\sum_{i=1}^{N} \frac{\tilde{u}_i - u_i}{\Delta t} = \sum_{i=1}^{N} s_i, \tag{3.9}$$

which is independent of α. The same result is obtained for the explicit scheme (3.7). The test problems considered here have $\sum_{i=1}^{N} s_i = 0$.

The linear system (3.8) for the one-dimensional problem requires the solution of a periodic tridiagonal matrix. This can be done directly, as in [12]. Here we will consider schemes based on a PJ approximation to the left-hand side of (3.8), i.e., the periodic tridiagonal system is replaced by a diagonal one.

Now consider the linear operator L^h. The relaxation operator S^h updates the error according to $\tilde{v}^h = S^h v^h$, where

$$S^h = I - \left(\tilde{L}^h\right)^{-1} L^h, \tag{3.10a}$$

and $\tilde{L}^h$ is an approximation to L^h that can be inverted easily. For the one-dimensional scalar problem studied here, we obtain

$$S^h = I - \beta(I - T^{-1}). \tag{3.10b}$$

For the explicit scheme (3.7), β equals the local Courant-Friedrichs-Lewy number σ:

$$\beta = \sigma \equiv \Delta t |a| / h, \tag{3.11a}$$

whereas for the PJ approximation to the implicit scheme (3.8)

$$\beta = \frac{\sigma}{1 + \alpha\sigma}. \tag{3.11b}$$

In both cases β can be considered as a relaxation parameter, which can be optimised to provide the best convergence factor.

We will now derive $\hat{S}^h$ for various relaxation schemes, in a form that obeys property (3.4). First consider schemes that do not couple frequencies, i.e., $\hat{s}_{21} = \hat{s}_{12} = 0$. The simplest is the PJ relaxation:

$$\hat{s}_{11}^{PJ} = 1 - \beta(1 - \hat{T}^{-1}). \tag{3.12a}$$

Stability requires

$$|\hat{s}_{11}^{PJ}(\theta)|^2 = 1 - 2\beta(1-\beta)(1 - \cos\theta) \le 1, \qquad 0 \le |\theta| \le \pi, \tag{3.12b}$$

implying

$$0 \le \beta \le 1. \tag{3.12c}$$

At the stability limit, $\beta = 1$ and $\hat{s}_{11}^{PJ} = \hat{T}^{-1}$. In that case, an explicit scheme runs at the maximum local time-step, and errors are convected over a distance h, the cell size. For a single-grid scheme, this choice is attractive, as errors are convected as fast as possible toward the boundaries (if present), where they leave the computational domain, or toward a shock, where they are absorbed by the dissipation in the shock.

The second relaxation scheme is a Multi-Stage (MS) Runge-Kutta scheme. Here we consider only two stages. The first stage is a PJ step with $\beta = \beta_1$, advancing the solution u to u'. The second stage resembles PJ, but now $\beta = \beta_2$ and the solution is advanced from u to $\tilde{u}$, using the residual $r' = r(u')$. The growth factor for one MS step is

$$\hat{s}_{11}^{MS} = 1 - \beta_2(1 - \hat{T}^{-1})[1 - \beta_1(1 - \hat{T}^{-1})], \tag{3.13a}$$

which is stable for

$$|\beta_1| \leq \tfrac{1}{2}, \qquad 0 \leq \beta_2 \leq 1 + 2\beta_1. \tag{3.13b}$$

At the stability limit $\beta_1 = \frac{1}{2}$, $\beta_2 = 2$, and $\mu = \hat{T}^{-2}$.

As a third relaxation scheme, we consider checkerboard or RB relaxation. It consists of a PJ sweep on the even points ($k = 0, 2, \cdots, N-2$), an update of the solution and residuals, followed by a similar operation on the odd points ($k = 1, 3, \cdots, N-1$):

$$\left.\begin{aligned} \tilde{v}_{2k} &= (1-\beta)v_{2k} + \beta v_{2k-1} \\ \tilde{v}_{2k+1} &= (1-\beta)v_{2k+1} + \beta\tilde{v}_{2k} \end{aligned}\right\} \quad k = 0, 1, \cdots, \tfrac{1}{2}N - 1. \tag{3.14}$$

Because of the asymmetric form of the CGC operator that is apparent in (3.6c), it is necessary to distinguish between two variants of RB. The distinction is based on the ordering with respect to the coarse-grid cell and the direction of the flow. The first variant will be noted by RB1 and corresponds to the (*even*, *odd*) ordering described in (3.14), i.e., first the cells with an even index and then those with an odd index are relaxed, assuming that $a > 0$ and that the restriction operator combines the cells $(2k, 2k+1)$. For negative a, first the odd and then the even cells are relaxed. The second version, RB2, has the opposite ordering. In other words, RB1 follows the flow, relative to the coarse-grid cell, and RB2 goes against the flow. We might call RB1 *Convective Red-Black* relaxation. For a single-grid scheme this distinction disappears.

In Fourier space we have, for $a > 0$,

$$\begin{aligned} \hat{s}_{11}^{RB1}(\hat{T}) &= 1 - \beta(1 - \hat{T}^{-1}) - \hat{s}_{12}^{RB1}(-\hat{T}), \\ \hat{s}_{12}^{RB1}(\hat{T}) &= -\tfrac{1}{2}\beta^2\hat{T}^{-1}(1 + \hat{T}^{-1}). \end{aligned} \tag{3.15a}$$

The other variant has

$$\hat{s}_{11}^{RB2}(\hat{T}) = \hat{s}_{11}^{RB1}(\hat{T}), \quad \hat{s}_{12}^{RB2}(\hat{T}) = -\hat{s}_{12}^{RB1}(\hat{T}). \tag{3.15b}$$

Both variants are stable for $0 \leq \beta \leq 1$.

Similar expressions can be derived for a second-order-accurate residual. In that case

$$\hat{L}^h = \frac{a}{h}(1 - \hat{T}^{-1})(1 + \tfrac{1}{4}(\hat{T} - \hat{T}^{-1})) \quad \text{for } a > 0. \tag{3.16}$$

With this residual, PJ as a single-grid relaxation scheme is unstable for all $\beta \neq 0$. The MS scheme is stable for $0 \leq \beta_1 \leq \frac{1}{2}$ and $0 \leq \beta_2 \leq 2\beta_1$, and RB for $0 \leq \beta \leq 1$. We can consider a multigrid scheme with either first-order or second-order accuracy on the coarser grids. In the latter case, an expression for the CGC operator $\hat{K}$ is obtained that is slightly more complicated than (3.6b). The values in Table 1 for second-order accuracy are based on such an operator.

3.4. Two-grid convergence factors. The performance of the two-grid method can be described by the smoothing rate of the relaxation scheme and by the two-grid convergence rate. The first measures how well the relaxation scheme removes the high frequency part of the error, and can be easily evaluated. The second measures convergence across the entire spectrum, under the assumptions that the coarser grid is solved exactly. Both rates will be considered in this section.

The smoothing rate is defined by

$$\overline{\mu} \equiv \max_{\pi/2 \leq |\theta| \leq \pi} \hat{s}_{11}(\theta), \tag{3.17}$$

for schemes without coupling, i.e., if $\hat{s}_{12} = \hat{s}_{21} = 0$. It describes how well the relaxation scheme removes the part of the spectrum that cannot be represented on the coarser grid. For schemes with coupling, such as RB, we still can use (3.17), assuming that there are no low frequencies, i.e., $\hat{v}(\theta) = 0$ for $0 \leq |\theta| < \pi/2$ (see [3, (3.2)]).

The two-grid convergence factor is

$$\overline{\lambda} \equiv \max_{0 \leq |\theta| \leq \pi} \rho(\hat{S}^{\nu_2} \hat{K} \hat{S}^{\nu_1}), \tag{3.18}$$

where $\rho(\cdot)$ is the spectral radius. It describes the result of applying ν_1 prerelaxation sweeps, a CGC, and ν_2 postrelaxation sweeps.

Table 1 lists smoothing rates and two-grid convergence rates for the three relaxation schemes of §3.3. The number of relaxation sweeps $\nu = \nu_1 + \nu_2 = 1$. The results have been obtained by evaluating (3.17) and (3.18) analytically or numerically. Minimum values with respect to the relaxation parameter(s) for a given relaxation scheme are marked by an asterisk. They have been computed by analytical or numerical optimisation of $\overline{\mu}$ and $\overline{\lambda}$, respectively, as a function of the relaxation parameter(s).

For the first-order discretisation, PJ with $\beta = \frac{1}{2}$ is obviously the best choice. Both the MS scheme and RB relaxation require more operations per sweep. The MS scheme reduces to PJ relaxation when optimised for $\overline{\lambda}$. It is clear that *optimising the smoothing rate does not necessarily imply a good two-grid convergence factor.* RB relaxation becomes dependent on the ordering when coarser grids are involved. For $\beta = 1$, the version that follows the flow (RB1) has a zero convergence factor, whereas RB2 is unstable.

It should be noted that PJ for a first-order-accurate residual and $\beta = \frac{1}{2}$ lets

$$\hat{K}\hat{S}^{PJ} = -\tfrac{1}{4}(\hat{T} - \hat{T}^{-1}) \begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix}, \tag{3.19}$$

resulting in a spectral radius zero. However, if PJ is applied more than once ($\nu > 1$), different results are obtained. For instance, $\overline{\lambda} = \frac{1}{2}$ if $\nu = 2$.

For Convective RB relaxation with $\beta = 1$ we have a stronger result than for PJ:

$$\hat{K}\hat{S}^{RB1} = \hat{S}^{RB1}\hat{K} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}. \tag{3.20a}$$

TABLE 1

Smoothing rate $\overline{\mu}$ and two-grid convergence factor $\overline{\lambda}$ for various relaxation schemes in the linear constant-coefficient case. The amount of underrelaxation is given by β. Minimum values with respect to the relaxation parameter(s) are denoted by an asterisk. The results are obtained for one relaxation step ($\nu = 1$). Second-order PJ *is unstable as a single-grid scheme.* RB1 *follows the flow, as seen from the coarse-grid cell, whereas* RB2 *acts in the opposite direction.*

Accuracy	First-order			Second-order		
Scheme	β	$\overline{\mu}$	$\overline{\lambda}$	β	$\overline{\mu}$	$\overline{\lambda}$
PJ	$\frac{1}{2}$	0.707*	0 *	$\frac{1}{5}$	0.949*	0.717
	$\frac{1}{2}$	0.707	0 *	0.145	0.953	0.711*
MS	$\frac{1}{3}$, 1	0.333*	0.333	0.350, 0.592	0.645*	0.677
	0, $\frac{1}{2}$	0.707	0 *	0.295, 0.588	0.719	0.517*
RB1	0.682	0.457*	0.101	0.510	0.806*	0.552
	1	0.707	0 *	0.928	1.153	0.386*
	$\frac{1}{2}$	0.530	0.25	0.405	0.821	0.596
RB2	0.682	0.457*	0.830	0.510	0.806*	0.676
	1	0.707	2	0.928	1.153	3.829
	$\frac{1}{2}$	0.530	0.25 *	0.405	0.821	0.636*

To obtain this performance in an actual computer code, we have to adapt the ordering to the direction of the flow. This can be avoided by choosing $\beta = \frac{1}{2}$, resulting in a convergence factor $\overline{\lambda} = \frac{1}{4}$. Alternatively, two relaxation sweeps with $\beta = 1$ can be applied, one with RB1 and one with RB2. Then

$$\hat{S}^{RB1}\hat{K}\hat{S}^{RB2} = \hat{S}^{RB2}\hat{K}\hat{S}^{RB1} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}. \tag{3.20b}$$

The behaviour of RB for $\beta = 1$ is better understood by considering (3.6c) and (3.14), assuming $a > 0$. After the CGC, $v_{2k-1} = 0$ and the first RB1 sweep at cell $(2k)$ convects this error from cell $(2k-1)$ to $(2k)$, thus making $\tilde{v}_{2k} = 0$. The second RB1 sweep at cell $(2k+1)$ is actually superfluous. On the other hand, RB2 selects $v_{2k} \neq 0$ and convects this error, resulting in the amplification of the shorter waves.

The fact that the second sweep of RB1 is superfluous can be exploited to construct a faster scheme, which we call Semi-Red-Black (SRB) relaxation. SRB only carries out the first step of RB1. Obviously, $\overline{\lambda} = 0$ for this scheme. However, for solutions with a sonic point or shock, SRB is likely to be less robust than RB1.

The most attractive scheme for a first-order-accurate discretisation is PJ with $\beta = \frac{1}{2}$ and $\nu = 1$. The convergence factors for second-order accuracy are not so good, as can be seen from Table 1. For a second-order discretisation, PJ is unstable as a single-grid scheme but still provides acceptable two-grid convergence factors.

In the numerical experiments of §4 only PJ relaxation with $\beta = \frac{1}{2}$ and RB1 are considered. Second-order-accurate solutions are computed by the Defect Correction technique [6, (14.3.1)]. The corresponding two-level operator is given by

$$I - (I - S^{\nu_2}KS^{\nu_1})(L^h_{p=1})^{-1}L^h_{p=2}, \tag{3.21}$$

where $L^h_{p=1}$ denotes the discrete residual operator of first-order accuracy (3.1) and $L^h_{p=2}$ of second-order accuracy (3.16). For PJ with $\beta = \frac{1}{2}$ and $\nu = \nu_1 + \nu_2 = 1$, we find $\overline{\lambda} = \frac{1}{2}\sqrt{3} = 0.866$, whereas RB1 with $\beta = 1$ provides $\overline{\lambda} = \frac{1}{2}$.

3.5. The singularity at the shock. The two-level analysis presented above describes the linear constant coefficient case. We might expect approximately the same two-grid convergence rates in the variable coefficient and nonlinear case, as long as the convection speed remains fairly constant with varying x and does not change sign. We will study this by numerical experiments for the nonlinear case in §4.2. The convection speed changes sign at a sonic point or shock. Convergence rates at the sonic point are considered in the Appendix, under the simplifying assumption that the convection speed is linear in x. Here we will examine the effect of the singularity at the shock on the convergence rate.

For Godunov's or Roe's scheme, which are identical around the shock, we have the following: if there is a shock, the two-grid convergence factor for a first-order scheme is in general $\overline{\lambda} \geq \frac{1}{2}$. To assert this, assume that we have two grids, a fine grid m with cell size h, and a coarse $m-1$ with cell size $2h$. Also assume that the coarse-grid problem is solved exactly, and that the residuals are small everywhere, even at the shock. The special situation of a steady shock positioned at the cell interface is excluded. In that case the errors v_k^h will be small, except at the shock, where v_M^h can still be large. Without loss of generality we choose an even index M for the cell that contains the singularity. The coarse-grid error, after restriction, is $v_{\frac{1}{2}M}^H = \frac{1}{2}(v_M^h + v_{M+1}^h) \simeq \frac{1}{2}v_M^h$. The exact solution of the coarse grid is prolongated back to the fine grid, resulting in

$$\begin{aligned} \tilde{v}_M^h &= v_M^h - v_{\frac{1}{2}M}^H \simeq \tfrac{1}{2}v_M^h, \\ \tilde{v}_{M+1}^h &= v_{M+1}^h - v_{\frac{1}{2}M}^H \simeq -\tfrac{1}{2}v_M^h. \end{aligned} \tag{3.22}$$

If the new $\tilde{u}_M^h$ still obeys

$$\tilde{u}_{M-1}^h > \tilde{u}_M^h > \tilde{u}_{M+1}^h \quad \text{with } \tilde{u}_{M-1}^h > 0 > \tilde{u}_{M+1}^h, \tag{3.23}$$

then only $\tilde{u}_{M+1}^h$ will result in fairly large residuals at M and $M+1$. An ideal relaxation scheme such as RB1 with $\beta = 1$ will first bring $\tilde{v}_{M+1}^h$ down to practically zero, causing the residuals at M and $M+1$ to vanish. The error $\tilde{v}_M^h \simeq \frac{1}{2}v_M^h$ will be unaffected. Thus the two-grid convergence factor $\overline{\lambda} = \frac{1}{2}$ in this case. In general, we expect $\overline{\lambda} \geq \frac{1}{2}$.

We can just accept the slower convergence at the shock. However, after every prolongation, there will be fairly large residuals near the shock. If these are not sufficiently damped by the relaxation scheme, convergence may be lost. PJ is likely to be good enough to obtain convergence. This will be further investigated by numerical experiments (§4).

The problem is that the singularity shows up during relaxation. As a remedy, we propose to use *local relaxation* (cf. [2]), with *local conservation* imposed as an extra condition to remove the singularity at the shock. In practice this is accomplished by selecting, say, four cells, the first having an even index (this determines its position with respect to the coarse-grid cell), and the second or third containing the shock. Let the indices of these cells be elements of the set $\mathcal{L}$. The SER method (3.8) is applied with $\alpha = 1$ and

$$\frac{1}{\Delta t} = \max_{k \in \mathcal{L}}(|r_k|/a_k^*). \tag{3.24}$$

Here a_k^* is an numerical approximation to the convection speed, to be specified in §4. The linear system associated with the SER scheme is singular at the shock for

$\Delta t \to \infty$. At the shock, we replace the equation for which $\partial r_M/\partial u_k = 0$, $k \in \mathcal{L}$, by local conservation:

$$\sum_{k\in\mathcal{L}} (\tilde{u}_k - u_k) = 0, \tag{3.25}$$

even for finite Δt. In the numerical experiments described in §4.3, it appeared to be sufficient to apply this once after every prolongation on every grid.

The condition of local conservation (3.25) can be justified as follows. Consider a sequence of cells u_l, $l = 1, \cdots, L$, which is a renumbered subset of the solution on the entire grid. Let one of these cells with index M, $1 < M < L$, contain the shock, and let u_l be positive left from the shock, and negative right from the shock. Here we have assumed that Godunov's or Roe's scheme is used. In addition, we assume that L is even and that cells $l = 1$ and $l = 2$ coincide with a cell on the coarser grid. The same is then true for the other pairs of cells corresponding to higher values of l. We may expect the average value of the solution directly after prolongation to be converged: $\sum_{l=1}^{L} u_l \simeq \sum_{l=1}^{L} \overline{u}_l$. Given this expression, (3.25) follows by requiring $\tilde{u}_l$ to equal the desired result $\overline{u}_l$.

If we want to avoid the replacement of the singular equation at the shock by (3.25), the SER scheme can be used without modification. This is due to the global conservation property (3.9). For the explicit scheme (3.7) and the implicit scheme (3.8) we have

$$\frac{1}{\Delta t}\sum_{l=1}^{L}(\tilde{u}_l - u_l) = \left(\sum_{l=1}^{L} s_l\right) - \tfrac{1}{2}(u_{L+1}^2 - u_0^2). \tag{3.26}$$

Note that u_0 and u_{L+1} are taken as fixed boundary values here, and that u_l is positive left and negative right from the shock. After prolongation, the right-hand side of (3.26) will have appeared on coarser grids as the restriction of the fine-grid residuals, or as the sum of restricted residuals. Convergence on coarser grids causes the sum of these residuals to vanish, at least approximately, which implies that directly after prolongation (3.26) is approximately the same as (3.25).

If the E–O, or FVS scheme, is used, local relaxation with local conservation can be carried out in a similar way.

4. Numerical experiments.

4.1. Technical details.

The numerical experiments are carried out in a FMG setting. The steady state on a grid with two cells is computed directly from the nonlinear equations, subject to (2.13), and then interpolated to the next finer grid to obtain a good initial guess. The corresponding interpolation operator Π_{2h}^{h} lets

$$u_{2k}^{h} = u_k^{2h} - \tfrac{1}{4}\Delta_k^{2h}, \qquad u_{2k+1}^{h} = u_k^{2h} + \tfrac{1}{4}\Delta_k^{2h}, \tag{4.1}$$

and is third-order-accurate if the solution is smooth. At the shock, however, an $O(1)$ error results. The numerical experiments described below show that this can be reduced to $O(h^p)$ for Godunov's or Roe's scheme by using local relaxation. The E–O, or FVS scheme, maintains an $O(1)$ error at the shock, both for $p = 1$ and $p = 2$. Still, local relaxation is carried out after grid refinement.

After interpolation by Π_{2h}^{h} (and local relaxation, if desired) the stationary solution is computed by a FAS scheme. As the multigrid strategy, we use three multigrid cycles with respect to each of the coarser grids to simulate a two-grid algorithm. For convergence down to the truncation error, an F-cycle [20] is chosen. In all cases, one postrelaxation sweep is carried out ($\nu_1 = 0$, $\nu_2 = 1$). On the coarsest grid ($N = 2$), the nonlinear equations are solved directly.

Damped PJ and RB1 are used as relaxation schemes. The first is implemented as

$$\tilde{u}_k = u_k + \beta(a_k^*)^{-1} r_k, \tag{4.2}$$

with $\beta = \frac{1}{2}$. The two steps of RB1 are carried out in the same way with $\beta = 1$. The nonlinear residual is updated after each of the two steps. We consider the following approximations to the convection speed:

$$a_k^* = -\frac{\partial r_k}{\partial u_k}, \tag{4.3a}$$

$$a_k^* = \max_l \left(-\frac{\partial r_k}{\partial u_k}, \frac{\partial r_k}{\partial u_l}\right), \tag{4.3b}$$

$$a_k^* = \tfrac{1}{2}\left(-\tfrac{\partial r_k}{\partial u_k} + \textstyle\sum_{l \neq k} \tfrac{\partial r_k}{\partial u_l}\right). \tag{4.3c}$$

The first choice (4.3a) cannot be applied with Godunov's or Roe's scheme, because of the singularity at the shock. The E–O, or FVS scheme, does not suffer from this drawback. The second and third choice can be used at the singularity.

If Godunov's, the E–O, or the FVS scheme is used, a_k^* may become small around the sonic point or at the shock, resulting in too large changes of the solution. To avoid this, a_k^* can be replaced by

$$\tilde{a}_k^* = a_k^* + |r_k|/a_k^*, \tag{4.4}$$

which is the equivalent of a SER scheme with a local time-step.

Local relaxation, if used, is carried out on a subset of four or six cells. These are determined by finding the index of the maximum residual and positioning it in the middle of the subset, in such a way that the first cell has an even index. If a singular equation is detected, it is replaced by the condition of local conservation (3.25).

4.2. Results for the smooth test problem. We have used the smooth problem (2.2) to make a comparison between two-grid convergence factors for the nonlinear equation and the linear two-level results. On coarser grids, three multigrid cycles were used to simulate a two-level algorithm with an exact solution of the coarse-grid equations [6, §2.5]. No local relaxation was applied. The experiments showed an $O(h)$ difference between the observed asymptotic convergence factors and the two-level results for the linear constant-coefficient case listed in Table 1. It is likely that close to the solution and away from sonic points and shocks, the nonlinear problem behaves as a linear one with a variable coefficient. The convection speed then has an $O(h)$ variation with x, which probably accounts for the observed $O(h)$ difference in the convergence factors.

The predicted convergence factors for the Defect Correction technique, given at the end of §3.4, were confirmed by the numerical experiments. Here the observed values tended to be slightly better.

The results were insensitive to the choices of a_k^* (4.3) or $\tilde{a}_k^*$ (4.4). There is no distinction between Godunov's, Roe's, and the E–O scheme for the current test problem, as u_k is positive on the entire grid.

For practical purposes, convergence down to the truncation error is sufficient. In that case, the experiments showed that a first-order solution could be obtained in one F-cycle per grid, using damped PJ or RB1 as part of a FMG scheme. A second-order solution computed with the Defect Correction technique required about four F-cycles per grid with damped PJ as relaxation scheme. A few more cycles were required for computing the steady state on the coarser grids ($N > 64$). RB1 required three F-cycles on the coarser, and two F-cycles on the finer grids (down to $N = 512$).

4.3. The discontinuous case. Several issues will be discussed for the discontinuous case (2.3): the choice of the discretisation, the effect of the various a_k^* (4.3) and $\tilde{a}_k^*$ (4.4), the singularity at the shock, and defect correction. We start with the discretisation scheme.

The experiments indicate that Godunov's scheme does not always provide proper convergence rates around the sonic point. This is partly explained by the analysis presented in the Appendix, which shows divergence for certain positions of the sonic point. Another complication occurs if $\overline{u}_k > 0$ is the steady state just right of the sonic point, and the initial guess $u_k < 0$. In that case u_k has to go through zero to reach the steady state. In more detail, the following happens.

Let the stationary sonic point lie between $\overline{u}_{k-1} < 0$ and $\overline{u}_k > 0$. Allowing for an initial guess $u_k < 0$, the residual $r_k = s_k - u_k|u_k|/(2h) = (\overline{u}_k^2 - u_k|u_k|)/(2h)$, and the corresponding value of $a_k^* = |u_k|/h$ by (4.3a) and (4.3b). This value is modified according to (4.4), and used in the iterative method (4.2). For $\beta = 1$ the right-hand side of (4.2) becomes $2u_k^3/(1 + 3u_k^2)$ if $u_k < 0$ and $\overline{u}_k = 1$. Thus, convergence is obtained to $u_k = 0$ rather than $\overline{u}_k = 1$. Once u_k has become zero, the iterations continue in a different way towards $\overline{u}_k = 1$.

As a whole, this process will take too long. For this reason, and because of the instability found in the Appendix, we abandon the use of Godunov's scheme in favour of Roe's (2.9) at the sonic point. In the following discussion, we will consider Roe's scheme (scheme I), and a variant of the E–O, or FVS scheme, that uses (2.9) at the sonic point (scheme II). Local relaxation is always applied several times after grid refinement. For scheme I, we use four cells for local relaxation. For scheme II, the experiments showed that four cells did not provide satisfactory results, but six cells allowed us to obtain convergence results comparable to scheme I.

If damped PJ without local relaxation was used, the convergence towards the steady state was dominated by the shock. For scheme I (Roe's) we observed a convergence factor 0.5 for the choices (4.3b) and (4.3c) of a^*, with or without the modification (4.4). This is in agreement with the discussion of §3.5. The two-grid convergence factor was obtained for a simulated two-grid algorithm, using three multigrid cycles on each of the coarser grids.

Scheme II provided values between 0.3 and 0.7, depending on the choice of a^* and the position of the shock. The choice (4.3a) did not work unless modified by (4.4), and even then the convergence was not very good. For (4.3b) with or without (4.4), we observed convergence factors between 0.3 and 0.6, and for (4.3c) between 0.4 and 0.5, depending on the position of the shock. Given the lower accuracy of scheme II, not much is gained by the sometimes slightly faster convergence.

If local relaxation was used once after every prolongation, the convergence rate was dominated by the slower convergence at the sonic point.

Next we consider convergence down to the truncation error. If no local relaxation was applied after prolongation, but only after grid refinement, both schemes I and II required two F-cycles with (4.3b) and three F-cycles with (4.3c). It turned out that with local relaxation, just one F-cycle was sufficient to obtain the steady state for both scheme I and II with damped PJ.

Second-order accurate solutions can be computed by the Defect Correction technique. For schemes I and II this required about eight F-cycles when damped PJ was used. Convergence for (4.3b) was slightly better than for (4.3c). For RB1 with (4.3b) and $\beta = 1$, a stationary solution was obtained in five F-cycles. Without local relaxation after prolongation, about 15 F-cycles were required for both schemes I and II. Here local relaxation was applied only after grid refinement. Thus, convergence was about a factor two slower if local relaxation after prolongation was omitted.

5. Conclusions. Convergence towards the steady state of the one-dimensional Burgers equation by means of a multigrid method has been studied. In order to exploit the global character of multigrid relaxation, only relaxation schemes that affect the solution locally have been considered. This excludes a global scheme such as line relaxation, and also Gauss–Seidel relaxation which uses only local data but has a global effect.

Linear two-level analysis has been carried out for a one-dimensional upwind differenced convection equation with a constant coefficient. Both smoothing rates and two-grid convergence factors have been derived. It turns out that optimising the smoothing rate does not necessarily imply a good two-grid convergence factor.

Experiments on the nonlinear inviscid Burgers equation show an $O(h)$ difference between the observed convergence factors for a simulated two-grid algorithm and the two-grid convergence factors for the linear constant-coefficient case, if there are no shocks or sonic points. The linear constant-coefficient case cannot explain what happens if the convection speed changes sign, i.e., at the sonic point and the shock. Convergence at the sonic point can be analysed separately.

At the shock the discrete nonlinear steady-state equations, obtained by upwind differencing, become singular. This reflects the singularity of the differential equation, which is overcome by invoking the jump condition across a discontinuity. The jump condition is automatically satisfied if the equation is solved in conservation form. In the present steady-state calculations, only conservation in space is retained. Conservation in time is dropped to obtain fast convergence. This introduces the singularity at the shock, and makes the steady state nonunique. The obvious way to regularise the discrete equations is by requiring global conservation in time, which is imposed as an additional condition on the coarsest grid. In this way, the multigrid method converges towards the correct steady state.

The linear two-level analysis provides a zero two-grid convergence factor for damped PJ. The numerical experiments confirm this, within $O(h)$, for a smooth test problem. At the shock, the convergence rate is about 0.5, which can be improved by local relaxation using a variant of Newton's method. Here the additional condition of local conservation is imposed to remove the singularity. If local relaxation was applied once after every prolongation, the steady state for the discontinuous test problem could be computed in only one F-cycle, given an initial guess from the coarser grid.

RB relaxation, if used as a single-grid scheme, is independent of the colouring, i.e., the convergence rate is the same if first the red and then the black cells are relaxed, or the other way around. This property is lost if RB is used as part of a multigrid scheme. The multigrid convergence rate depends on the colouring relative to the coarse-grid cells and the direction of the flow. If the order in which first the red and then the black cells are relaxed follows the flow as seen from each coarse-grid cell, then RB has a convergence rate zero in the linear constant-coefficient case; otherwise, it is unstable. A damped version of RB does not suffer from the instability.

Steady solutions with second-order accuracy have been computed by the Defect Correction technique. Convergence is considerably slower than in the first-order case.

An extension of the present work to two dimensions can be found in [15]. Nonlinear singularities of the discrete equations are not considered in that paper. In two or three dimensions, such singularities are less likely to occur, because the coupling between the state variables is increased by having more than one direction. Still, a regularisation is required if the nonlinear steady-state problem is singular. The regularisation chosen here is based on global conservation in time of the initial states. However, this approach cannot be used if singularities occur in more than one cell. In [13], the isenthalpic Euler equations were solved, and some time accuracy was maintained, which avoids possible problems with singularities. This may not be sufficient in all cases. An alternative regularisation can be obtained through additional viscous terms, at the expense of the spatial accuracy. Brandt proposes the use of double discretisation [3, §10.2] to overcome this problem, but it remains to be seen if a robust method can be obtained by this technique.

Appendix. Convergence rates at the sonic point. Two-grid convergence rates at the sonic point are estimated by assuming the numerical steady state to be linear in x, and by linearising the discrete nonlinear equations. We consider six cells and a numerical steady state

$$\text{(A1)} \qquad \overline{u}_k = a_k = a_2 + (k-2)ha_2' + O(h^2) \quad \text{with } a_2' > 0, \qquad k = 0, \cdots, 5.$$

There are two cases, corresponding to two different positions of the sonic point with respect to the coarse-grid cells:

$$\text{(A2a)} \qquad \text{Case 1}: \quad a_2 = \alpha h a_2',$$

$$\text{(A2b)} \qquad \text{Case 2}: \quad a_2 = -\alpha h a_2',$$

where $0 \le \alpha \le 1$. The numerical sonic point lies between cell one and two in Case 1, and between two and three in Case 2. To obtain a linear discrete operator, we evaluate the nonlinear residuals, rewrite them in terms of the iteration error $v_k = \overline{u}_k - u_k$, and ignore terms of $O(v_k^2)$. The upwind differencing of the nonlinear equation is based on $\overline{u}_k$, not on u_k. Note that these two choices *cannot* be justified by assuming v_k to be small. Their only justification is the simplification of the analysis.

We will not present all the details of the derivation of the two-grid convergence factors, but merely mention some of the intermediate results for Case 1 when using Roe's scheme. In that case, we have, for instance,

$$\text{(A3a)} \qquad \begin{aligned} r_0 &= s_0 - \frac{1}{2h}(u_1^2 - u_0^2) = \frac{1}{2h}(\overline{u}_1^2 - \overline{u}_0^2) - \frac{1}{2h}\left((\overline{u}_1^2 - v_1)^2 - (\overline{u}_0 - v_0)^2\right) \\ &\simeq (a_1 v_1 - a_0 v_0)/h, \end{aligned}$$

and

$$\text{(A3b)} \qquad \begin{aligned} r_2 &= s_2 - \frac{1}{2h}(u_2^2 - u_1 u_2) \\ &= \frac{1}{2h}(\overline{u}_2^2 - \overline{u}_1\overline{u}_2) - \frac{1}{2h}\left((\overline{u}_2 - v_2)^2 - (\overline{u}_1 - v_1)(\overline{u}_2 - v_2)\right) \\ &\simeq \left(a_2 v_2 - \tfrac{1}{2}(a_1 v_2 + a_2 v_1)\right)/h. \end{aligned}$$

The other residuals follow in a similar way by using Roe's scheme and choosing the upwind direction from $\overline{u}$. In this way a linear operator L^h is obtained, which can be written as a 6×6 matrix acting on $(v_0, v_1, \cdots, v_5)^T$.

Only damped PJ is considered as a relaxation operator. For the choice (4.3a) of a_k^*, this relaxation operator becomes

$$S^{PJ} = I - \tfrac{1}{2} N^{-1} L^h, \tag{A4}$$

where N is a diagonal matrix with its diagonal elements equal to those of L^h. The restriction and prolongation operators are chosen as in §3.2. For the CGC operator, we need L^H, which can be obtained by Galerkin coarsening ($L^H = I_h^H L^h I_H^h$), or evaluated directly from the coarse-grid equations, using $\overline{u}_k^H = a_k^H = \frac{1}{2}(a_{2k}^h + a_{2k+1}^h)$ for $k = 0, 1, 2$. Only the last option is considered here. In Case 1 with Roe's scheme, we obtain

$$L^H = \frac{1}{2h} \begin{pmatrix} \frac{1}{2}a_1^H - a_0^H & \frac{1}{2}a_0^H & 0 \\ -\frac{1}{2}a_1^H & a_1^H - \frac{1}{2}a_0^H & 0 \\ 0 & -a_1^H & a_2^H \end{pmatrix}. \tag{A5}$$

Similar results are obtained for Case 2, and for Godunov's scheme. It should be noted that in Case 2 we obtain operators that are different for $\alpha \leq \frac{1}{2}$ and $\alpha \geq \frac{1}{2}$.

Once the relaxation operator S^{PJ} and the CGC operator K have been obtained, the two-grid convergence rate follows from

$$\overline{\lambda}(\alpha) = \rho(K S^{PJ}). \tag{A6}$$

Table 2 lists the smallest and largest values of the two-grid convergence factor, i.e., $\min_\alpha \overline{\lambda}(\alpha)$ and $\max_\alpha \overline{\lambda}(\alpha)$. Different results are obtained for Godunov's and Roe's scheme, and for the choices of a_k^* listed in (4.3). The entry ∞ is due to a vanishing coarse-grid convection speed a_1^H for $\alpha = \frac{1}{2}$ in Case 2. This problem does not occur for Roe's scheme.

TABLE 2

Two-grid convergence factors around the sonic point. The values shown are the minimum and maximum of $\overline{\lambda}(\alpha)$ over α, which determines the position of the sonic point relative to the grid. The choices of a_k^* are listed in (4.3). *Damped* PJ ($\beta = 1/2$) *is used as relaxation scheme.*

Scheme	a_k^*	Case 1		Case 2	
Godunov	a, b	0.354	0.500	0.500	∞
	c	0.500	0.556	0.375	0.556
Roe	a, b	0.500	0.500	0.318	0.359
	c	0.249	0.312	0.375	0.614

REFERENCES

[1] W. K. ANDERSON, *Implicit multigrid algorithms for the three-dimensional flux split Euler equations*, Ph. D. thesis, Mississippi State University, Mississippi State, MS, 1986.

[2] D. BAI AND A. BRANDT, *Local mesh refinement multilevel techniques*, SIAM J. Sci. Statist. Comput., 8 (1986), pp. 109-134.

[3] A. BRANDT, *Guide to multigrid development*, Lecture Notes in Math., 960 (1981), pp. 220-312.

[4] B. ENGQUIST AND S. OSHER, *Stable and entropy satisfying approximations for transonic flow calculations*, Math. Comp., 34 (1980), pp. 45-75.

[5] S. K. GODUNOV, *Finite-difference method for numerical computation of discontinuous solutions of the equations of fluid dynamics*, Mat. USSR Sb., 47 (1959), pp. 271-306.

[6] W. HACKBUSCH, *Multi-Grid Methods and Applications*, Springer Series in Computational Mathematics, 4, Springer-Verlag, Berlin, Heidelberg, New York, 1985.

[7] P. W. HEMKER AND S. P. SPEKREIJSE, *Multiple grid and Osher's scheme for the efficient solution of the steady Euler equations*, Appl. Numer. Math., 2 (1986), pp. 475-493.

[8] P. W. HEMKER, *Defect correction and higher order schemes for the multigrid solution of the steady Euler equations*, Lecture Notes in Math., 1228 (1986), pp. 149-165.

[9] A. JAMESON, *Solution of the Euler equations for two-dimensional transonic flow by a multigrid method*, Appl. Math. Comput., 13 (1983), pp. 327-356.

[10] D. C. JESPERSEN, *Design and implementation of a multigrid code for the Euler equations*, Appl. Math. Comput., 13 (1983), pp. 357-374.

[11] B. KOREN, *Defect correction and multigrid for an efficient and accurate computation of airfoil flows*, J. Comput. Phys., 77 (1988), pp. 183-206.

[12] W. A. MULDER AND B. VAN LEER, *Experiments with implicit upwind methods for the Euler equations*, J. Comput. Phys., 59 (1985), pp. 232-246.

[13] W. A. MULDER, *Multigrid relaxation for the Euler equations*, J. Comput. Phys., 60 (1985), pp. 235-252.

[14] ———, *Computation of the quasi-steady gas flow in a spiral galaxy by means of a multigrid method*, Astron. and Astrophys., 156 (1986), pp. 354-380.

[15] ———, *Analysis of a multigrid method for the Euler equations of gas dynamics in two dimensions*, in Multigrid Methods: Theory, Applications, and Supercomputing, S. McCormick, ed., Marcel Dekker, New York, 1988, pp. 467-489.

[16] R. H. NI, *A multiple grid scheme for solving the Euler equations*, AIAA J., 20 (1982), pp. 1565-1571.

[17] S. OSHER AND F. SOLOMON, *Upwind difference schemes for hyperbolic systems of conservation laws*, Math. Comput., 38 (1982), pp. 339-374.

[18] P. L. ROE, *Approximate Riemann solvers, parameter vectors, and difference schemes*, J. Comput. Phys., 43 (1981), pp. 357-372.

[19] J. STEGER AND R. WARMING, *Flux vector splitting of the inviscid gas dynamics equations with applications to finite-difference methods*, J. Comput. Phys., 40 (1981), pp. 263-293.

[20] K. STÜBEN AND U. TROTTENBERG, *Multigrid methods: Fundamental algorithms, model problem analysis and applications*, Lecture Notes in Math., 960 (1981), pp. 1-176.

[21] G. D. VAN ALBADA, B. VAN LEER, AND W. W. ROBERTS, *A comparative study of computational methods in cosmic gas dynamics*, Astron. and Astrophys., 108 (1982), pp. 76-84.

[22] B. VAN LEER, *Towards the ultimate conservative difference scheme. IV. A new approach to numerical convection*, J. Comput. Phys., 23 (1975), pp. 276-299.

[23] ———, *On the relation between the upwind-differencing schemes of Godunov, Enquist–Osher, and Roe*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 1-20.

[24] B. VAN LEER AND W. A. MULDER, *Relaxation methods for hyperbolic conservation laws*, in Numerical Methods for the Euler Equations of Fluid Dynamics, F. Angrand, A. Dervieux, J. A. Desideri, and R. Glowinski, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1985, pp. 312-333.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 1, pp. 51-62, January 1990

MULTIGRID METHODS FOR LOCATING SINGULARITIES IN BIFURCATION PROBLEMS*

SHLOMO TA'ASAN†

Abstract. This paper deals with multigrid methods for locating singular points for nonlinear equations, such as limit points and bifurcation points (perfect or imperfect), and is restricted to the self-adjoint case. A minimization problem that defines singular points is formulated. It treats uniformly limit points and bifurcation points unlike other methods that are designed to solve for one of the two. So it is particularly useful when the type of singularity, or even its existence, is not known in advance. Efficient multigrid methods for locating singular points based on the minimization problem are described. They solve the problems to the level of discretization errors in just a few work units (about 10 or fewer), where a work unit is the work involved in one local relaxation on the finest grid.

Key words. multigrid, singularities, bifurcation points, limit points

AMS(MOS) subject classifications. 65N20, 65C99, 65D99

1. Introduction. This paper discusses the detection of singular points as part of a continuation process and its accurate location once it has been detected. Generally, a nonlinear equation with a parameter is given, and the behavior of the solution as a function of the parameter is required. This may include, for example: (1) continuation along solution curves; (2) detection of singularities in the marching process; (3) locating singular points.

Continuation methods in which the problem is solved for one choice of the parameter and then changing it until a prescribed value is reached are commonly used. Such methods may encounter difficulties in cases where the problem becomes singular for some choice of that parameter. Different continuation techniques therefore have been developed [K], [M]. In the *arclength continuation* methods [DK], [K], [M], a new parameter, which represents an arclength along a solution curve, together with an extra equation, is introduced. This makes the continuation process very robust at regular and at limit points.

Since singularities play an important role in physical systems, their detection as part of the marching process may be required. This paper proposes a detection method based on monitoring (on the coarsest level in a multigrid algorithm) the eigenvalue that is the closest to zero. If a small (precise meaning is given later) extremum is detected in the behavior of the eigenvalue, further checks are done to ascertain the existence of a singular point of the continuous level in that vicinity. In case the extremum approaches zero, as the discretization parameter tends to zero, a singularity has been detected. The advantage of this method is that it can detect singularities of the continuous system that disappear on discretization.

Once a singularity is detected, its exact location may be needed. Several approaches for locating singularities have been developed by different authors [Moo], [MS], [BK], [R]. A different method is suggested here that uses a minimization problem to define

* Received by the editors July 20, 1987; accepted for publication (in revised form) November 23, 1988. This research was supported in part by the National Aeronautics and Space Administration under NASA contract NAS1-18107 while the author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, Virginia 23665 and in part by the U.S. Air Force Office of Scientific Research under grant AFOSR-86-127.

† Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot 76100, Israel.

the singular point. That problem is being solved directly, without involving iterations within iterations as other methods require. This leads to very efficient algorithms for locating singular points. The importance of this becomes clear when curves (in the parameter space) of singular points (envelope of singular points) are required. Such envelopes of singular points can be defined as solutions of a new minimization problem. A continuation method can then be applied to this new problem, resulting in significant savings in the computational work. In case of simple limit points, the minimization problem described above reduces to a set of equations treated in [MS]. The resulting system is regular in the vicinity of a simple limit point (see [MS]) and standard methods can be used for solving it.

Multigrid methods [B], [HT], [ST] can interact very naturally with the problems mentioned. Continuation in order to get to a vicinity of a singularity can be done mostly on coarse levels with refinement being done only when a singularity has been detected. Fine grid location of a singularity is found by an FMG algorithm that starts solving the problem on coarse grids and uses a fixed number of multigrid cycles per refinement.

The multigrid cycling on each level involves a relaxation that combines local and global steps. The local step is a standard one and serves to smooth the error in local quantities. The global step accelerates the convergence of the singular components in the solution that are slow to converge in local processes. While the local step is employed on all levels, the global steps are performed on the appropriate level (explained later).

Section 2 describes the basic tools such as an arclength continuation method, and the detection and location of singularities from solution of appropriate minimization problems. Section 3 describes multigrid algorithms for detecting and locating singular points. In § 4 the effectiveness of some of the algorithms presented in the paper is demonstrated. It is shown that limit points are located to the level of discretization errors using the 1-FMG Algorithm for an appropriate system of equations defining these points. Similar results using the 2-FMG Algorithm are obtained when bifurcation points are to be located.

2. Basic tools.

2.1. An arclength continuation method. Consider a nonlinear problem in the form

$$L(u,\lambda)=0 \tag{2.1}$$

where $L:\mathscr{H}\times\mathscr{R}\to\mathscr{H}$, $\mathscr{R}$ is the set of real numbers and $\mathscr{H}$ is some Hilbert space and such that $L_u(u,\lambda)$ is a self-adjoint operator. Smooth branches of solutions

$$\{\Gamma_{ab}: [u(s),\lambda(s)],\ s_a\leqq s\leqq s_b\} \tag{2.2}$$

are required, where $u(s)\in\mathscr{H}$, $\lambda(s)\in\mathscr{R}$. Here the parameter s is arbitrary, and we refer to it later as the arclength parameter.

The standard approach of using λ as the continuation parameter encounters difficulties when L_u becomes singular. A method that circumvents such difficulties to some extent is the *arclength continuation*. In this method (2.1) is replaced by

$$L(u(s),\lambda(s))=0, \tag{2.3}$$

$$N(u(s),\lambda(s),s)=0 \tag{2.4}$$

where $N:\mathscr{H}\times\mathscr{R}\times\mathscr{R}\to\mathscr{R}$ and $s\in\mathscr{R}$ is the independent parameter on the arc of solutions [K], [M]. Several choices for N are possible. Throughout this paper N is defined by

$$\begin{aligned}N(u(s),\lambda(s),s)\equiv{}&\theta\langle u-u_0,u_0-u_1\rangle+(1-\theta)(\lambda-\lambda_0)(\lambda_0-\lambda_1)\\&-(s-s_0)\sqrt{\theta\|u_1-u_0\|^2+(1-\theta)|\lambda_1-\lambda_0|^2}\end{aligned} \tag{2.5}$$

where s_0 and s_1 are two previous points on the branch of solutions, $u \equiv u(s)$, $u_0 \equiv u(s_0)$, $u_1 \equiv u(s_1)$, $\lambda_1 \equiv \lambda(s_1)$, $0 \leqq \theta \leqq 1$ and $\langle \cdot, \cdot \rangle$ denotes the inner product in $\mathcal{H}$. This definition of N is an approximation to a choice given in [K] and does not require the computation of the tangent to the solution curve.

The role of θ, in the definition of N, is to enable inexpensive marching through limit points where the curve has a high curvature. Note that the geometric meaning of (2.5) is that the angle between two successive changes in the solution, in the (u, λ) plane, is less than $\pi/2$ (assuming $s > s_0$). In case the stepsize is too large, for example, in encountering an angular limit point, this may not be the case, and the augmented system (2.3)-(2.4) may not have a solution unless s is close enough to s_0. With a proper choice of θ (usually either $\theta = 1$ or $\theta = 0$) marching through such points is possible, without decreasing the stepsize. This is possible since changing θ is related to a change in the curvature of the solution branch. Treating a limit point in λ, for example, becomes trivial with the choice $\theta = 1$, since with this parametrization the problem is regular.

A CONTINUATION PROCESS. For completeness of the presentation, a full continuation process is described here. Let $\theta = 0.5$.

Step 1.

- Set $s_0 = 0$, $\lambda = \lambda_0$, $u_0 = 0$.
- Solve $L(u_0, \lambda) = 0$, for u_0, keeping λ fixed.

Step 2.

- Set $\lambda = \lambda_1$, $u_1 = u_0$.
- Solve $L(u_1, \lambda) = 0$, for u_1, keeping λ fixed.
- Compute s_1 from the equation

$$(s_1 - s_0)^2 = \theta \|u_1 - u_0\|^2 + (1-\theta)|\lambda_1 - \lambda_0|^2.$$

Further Continuation Steps.

- Choose Δs (for example, $s_1 - s_0$). Set $s = s_1 + \Delta s$ and

$$u = u_1 + \frac{s - s_1}{s_1 - s_0}[u_1 - u_0]$$

$$\lambda = \lambda_1 + \frac{s - s_1}{s_1 - s_0}[\lambda_1 - \lambda_0].$$

- Solve the following equation for (u, λ)

$$L(u, \lambda) = 0$$

$$N(u, \lambda, s) = 0.$$

- Set $s_0 = s_1$, $s_1 = s$, $u_0 = u_1$, $u_1 = u$.

2.2. Detection of singularities. A singular point is a point (u_*, λ_*) on a branch of solutions for which $L_u(u_*, \lambda_*)$ is singular. A common way for detecting singularities is to check for a sign change in the determinant of the linearized system [K]. This, however, cannot detect bifurcation from eigenvalues of even multiplicity. Also it cannot detect a bifurcation of the differential equation that has become an imperfect bifurcation on the discrete level, since the determinant may not change sign along discrete solution curves in such cases. In fact, singularities may disappear on discretization (see [BRP], [Moo], [KK]).

In this paper a different approach is taken. Let (u_*, λ_*) be a singular point, and let φ be a corresponding singular eigenfunction, i.e.,

$$L_u(u_*, \lambda_*)\varphi = 0. \tag{2.6}$$

In the vicinity of the point (u_*, λ_*)

$$\mu^2(s) \equiv \min_{\varphi \neq 0} \left[\frac{\langle L_u(u(s), \lambda(s))\varphi, \varphi\rangle}{\langle \varphi, \varphi\rangle} \right]^2 \quad \text{subject to} \tag{2.7}$$

$$L(u(s), \lambda(s)) = 0, \tag{2.8}$$

is close to zero, and it vanishes at $(u, \lambda) = (u_*, \lambda_*)$.

Detecting singular points is done by computing approximately $\mu^2(s)$ on the coarsest level. Once a minimum has been found, further checks are needed to decide whether a zero for $\mu(s)$ exists on the continuous level. For the continuous level the extremum of $\mu(s)^2$ is zero at singular points, while it may not be so on the discrete level. However, it approaches zero as the discretization parameter goes to zero.

We refer the reader to some standard references, e.g., [SW], [Moo], for definition of limit points, bifurcation points, and imperfect bifurcation points.

2.3. Locating singularities. Having detected the possibility of a singular point using the coarsest level, further checks (to confirm the existence of a singularity) and its exact location may be required. In the vicinity where $\mu(s)^2$ has a minimum, the following minimization problem may be used to accurately locate a singularity:

$$\min_{(u,\lambda,\varphi)} \mu^2 \quad \text{subject to} \tag{2.9}$$

$$L(u, \lambda) = 0, \tag{2.10}$$

$$L_u(u, \lambda)\varphi = \mu\varphi, \tag{2.11}$$

$$\|\varphi\|^2 = 1. \tag{2.12}$$

The problem is solved on the coarsest levels. If $\mu(s) \not\to 0$, as the meshsize decreases, the process of solving (2.9)-(2.12) is stopped and the original continuation process continues.

This minimization problem can serve for locating either limit points or bifurcation points. Together with $\mu^2(s)$, $\langle L_\lambda, \varphi\rangle$ is also needed to decide about the nature of the singularity located, i.e., whether it is a limit point, a bifurcation point, or an imperfect (discrete) bifurcation. At bifurcation points $\langle L_\lambda, \varphi\rangle = 0$, and at limit points $\langle L_\lambda, \varphi\rangle \neq 0$, while at both $\mu(s) = 0$. When a discretization is used to approximate a singular point, the two quantities μ_*^h and $\langle L_\lambda^h, \varphi^h\rangle$ are available. The nature of a singularity is determined as follows:

```
if $\mu_h^* \not\to 0$,  as  $h \to 0$  then no singularity exists
else
  if $\langle L_\lambda^h, \varphi^h\rangle \to 0$  as $h \to 0$, a possible bifurcation point has been detected
  else a limit point has been detected.
```

The distinction between different types of singularities is done based on inner products involving higher derivatives of $L(u, \lambda)$ (see, e.g., [SW], [Moo]).

The location of the singular point, if detected, is obtained from the limit of the sequence (u_*^h, λ_*^h) that minimizes the discrete analogue of (2.9)-(2.12). Note that the decision about the existence of a singular point can be made in general only by solving a sequence of discrete problems, since a perfect bifurcation may become an imperfect one on discretization.

Sometimes it is known in advance that only limit points are involved. In such cases they can be located using the following system of equations, once the vicinity of the limit point is reached:

$$L(u, \lambda) = 0, \tag{2.13}$$

$$L_u(u, \lambda)\varphi = 0, \tag{2.14}$$

$$\|\varphi\|^2 = 1. \tag{2.15}$$

Note that the minimization problem described above reduces to (2.13)-(2.15) for the case of limit points. In a vicinity of a limit point the above system is regular (see [MS]).

3. Multigrid algorithms. As explained before, detecting a singularity begins with a continuation process that reaches the neighborhood of the singularity. In our numerical experiments a multigrid continuation algorithm was used. It is similar to the one used in [BK] with a few differences. One is the choice of N in (2.4). The other is the way the arclength equation was used. In [BK] it was used on the coarsest level only, while here it was used on the finest level with the FAS formulation on coarse grids. Another difference is the way the coarsest grid equations are being solved. In [BK] a Newton iteration was performed while here it was done using a local relaxation together with a global step to accelerate it. This idea can be used in case the Newton method does not work, for example, if the linearized system is singular (see the algorithm for locating singularities). Because of the similarity of our algorithm to the one in [BK] we will not describe it in detail here. A detailed description can be found in [T].

3.1. Coarse grid detection of singularities. The continuation algorithm is combined with a simple and inexpensive detection of singularities basically done on the coarsest level.

Actually what we compute on the coarsest level are the nearly singular eigenfunctions of L_u, and their corresponding eigenvalues. The device for that is Kaczmarz relaxation. It has the property of damping the error components that belong to eigenfunctions with relatively larger eigenvalues (in magnitude). The ones that correspond to nearly singular components are the slowest to converge. By starting with a random guess $\tilde{W}$ and relaxing the equation

$$L_u(u, \lambda)W = 0 \tag{3.1}$$

enough times until the convergence rate becomes very slow, the approximation $\tilde{W}$, to the actual solution $W = 0$, lies mainly in the subspace of nearly singular functions. Starting with this as an approximation we can solve the following eigenvalue problem:

$$L_u(u, \lambda)W - \mu W = 0, \tag{3.2}$$

$$\|W\|^2 = 1 \tag{3.3}$$

in order to define a function in the almost singular subspace. Assume for the moment that this subspace is of dimension one. A relaxation for that purpose consists of local and global steps. The local one is a Kaczmarz relaxation. The global one is of the following form:

$$W \leftarrow W\beta, \tag{3.4}$$

$$\mu \leftarrow \frac{\langle L_u(u, \lambda)W, W\rangle}{\langle W, W\rangle} \tag{3.5}$$

where β is such that the norm requirement is satisfied.

In the event that the nearly singular subspace is of a higher dimension than one, that algorithm will converge slowly very soon. In that case new eigenfunctions must be introduced into the process, where at each time the initial approximation for such an eigenfunction is taken from the residuals of the problems already in process that have been orthogonalized to previous eigenfunctions already found. The process should also include a Ritz projection as in [BMR] for that set of functions. In this way an orthogonal set of functions that spans the nearly singular subspace is found. Each of these solves a system such as (3.2)-(3.3) with a possibly different μ.

Once the minimal eigenvalue (in magnitude) is found, it is compared to the one found in the last two continuation steps to look for a minimum. Note that since this entire process is performed on the coarsest level, it is very inexpensive.

3.2. Multigrid method for locating singularities.

3.2.1. Limit points. Once a limit point has been detected by observing the behavior of $\lambda(s)$, for example, it may be required to actually locate it. This is done by applying an FMG algorithm to a discrete version of (2.13)-(2.15) once the continuation process has reached the neighborhood of the limit point. The relaxation process is described first. The rest is a standard FMG method for equations (2.13)-(2.15) using the FAS formulation.

Relaxation. A local process is used to smooth the error in both u and φ by relaxing the discrete version of (2.13) for u (keeping λ fixed), and (2.14) for φ (keeping (u, λ) fixed). This local step is employed on all levels. On the coarsest level an additional step is performed. It begins by updating the norm of φ by multiplying it by a proper constant, to satisfy the norm condition on this level, followed by

$$u \leftarrow u + \beta\varphi, \qquad \lambda \leftarrow \lambda + \delta \tag{3.6}$$

where (β, δ) satisfies

$$\langle L(u+\beta\varphi, \lambda+\delta), \varphi\rangle = \langle F, \varphi\rangle, \tag{3.7}$$

$$\langle L_u(u+\beta\varphi, \lambda+\delta)\varphi, \varphi\rangle = \langle G, \varphi\rangle. \tag{3.8}$$

Here F and G are the right-hand sides for the corresponding coarsest grid equations, respectively. The solution of (3.7)-(3.8) is done using a linearization and is justified since the system (2.13)-(2.15) is nonsingular in a vicinity of a limit point. Note that since the operator L_u is singular at a limit point, a local relaxation will be slowly converging for the φ component in u. This, however, is taken care of by the global step that changes u exactly in the direction of slow convergence.

The full description of the cycling algorithm is described next.

MULTIGRID ALGORITHM MGLLP. Consider a sequence of grids $\Omega_k (k \leqq M)$ with mesh sizes h_k satisfying $2h_{k+1} = h_k$. Suppose on each grid operators (L^k, L^k_u) are given in such a way that (L^k, L^k_u) $(k < M)$ is an approximation to (L^{k+1}, L^{k+1}_u) and the Ω_k grid equations are

$$L^k(\tilde{u}, \lambda) = F^k, \tag{3.9}$$

$$L^k_u \tilde{\varphi}^k = G^k, \tag{3.10}$$

$$\|\tilde{\varphi}\|^2_k = H^k. \tag{3.11}$$

Assume also that interpolation operators I^k_{k-1}, from coarse to fine grids, and restriction operators I^{k-1}_k, $\bar{I}^{k-1}_k$, from fine to coarse grids, are given.

Given an approximate solution $(u^k, \varphi^k, \lambda)$ to (3.9)-(3.11), the multigrid cycle for improving it is denoted by

$$(u^k, \varphi^k, \lambda) \leftarrow \text{MGLLP}(k, u^k, \varphi^k, \lambda, F^k, G^k, H^k) \tag{3.12}$$

and is defined recursively as follows:

if $k=1$ **then** solve (3.9)-(3.11) by enough relaxations (to achieve a desired accuracy)
else

- Perform ν_1 relaxation sweeps on (3.9)-(3.11), starting with $(u^k, \varphi^k, \lambda)$ and resulting in a new approximation $(\bar{u}^k, \bar{\varphi}^k, \bar{\lambda})$.
- Starting with $u^{k-1} = \bar{I}_k^{k-1}\bar{u}^k$, $\varphi^{k-1} = \bar{I}_k^{k-1}\bar{\varphi}^k$ make γ successive cycles of the type

  $$(u^{k-1}, \lambda^{k-1}, \lambda) \leftarrow \text{MGLLP}(k-1, u^{k-1}, \varphi^{k-1}, \lambda, F^{k-1}, G^{k-1}, H^{k-1})$$

  where

  $$F^{k-1} = I_k^{k-1}(F^k - L^k\bar{u}^k) + L^{k-1}\bar{I}_k^{k-1}\bar{u}^k$$
  $$G^{k-1} = I_k^{k-1}(G^k - L_u^k\bar{\varphi}^k) + L_u^{k-1}\bar{I}_k^{k-1}\bar{\varphi}^k$$
  $$H^{k-1} = (H^k - \|\bar{\varphi}^k\|_k^2) + \|\bar{I}_k^{k-1}\bar{\varphi}^k\|_{k-1}^2.$$
- Calculate $\bar{u}^k = \bar{u}^k + I_{k-1}^k(u^{k-1} - \bar{I}_k^{k-1}\bar{u}^k)$, $\bar{\varphi}^k = \bar{\varphi}^k + I_{k-1}^k(\varphi^{k-1} - \bar{I}_k^{k-1}\bar{\varphi}^k)$.
- Perform ν_2 relaxation sweeps on (3.9)-(3.11) starting with $\bar{u}^k$, $\bar{\varphi}$, λ and yielding $(u^k, \varphi^k, \lambda)$, the final result of (3.12).

FMGLLP ALGORITHM. To obtain full efficiency, the first approximation on a given level is obtained from a solution of the same problem on the next coarser level, which itself has been calculated in a similar way. The resulting algorithm is called (FMGLLP) and is described next.

Let Π_{k-1}^k be an interpolation operator (usually of higher order than I_{k-1}^k). Given the problem (3.9)-(3.11) with $k=M$, the N-FMG solution of that problem is:

Initial setup.
Set $F^M=0$, $G^M=0$, $H^M=1$.
for $k=M-1, \cdots, 1$ **do**:
- $F^k = I_{k+1}^k F^{k+1}$, $G^k = I_{k+1}^k G^{k+1}$, $H^k = H^{k+1}$.

N-FMG ALGORITHM.
Calculate u^1 the solution of (3.9)-(3.11) for $k=1$ by several relaxations.
for $k=2, \cdots, M$ **do**:
- Calculate $u^k \leftarrow \Pi_{k-1}^k u^{k-1}$, $\varphi^k \leftarrow \Pi_{k-1}^k \varphi^{k-1}$.
- Perform the cycle $(u^k, \varphi^k, \lambda) \leftarrow \text{MGLLP}(k, u^k, \varphi^k, F^k, G^k, H^k)$ N times.

The effectiveness of the multigrid algorithm for locating limit points is demonstrated in § 4.1, where the limit point for the Bratu problem is computed to the level of discretization errors with the above algorithm using $N=1$, i.e., the 1-FMG algorithm.

3.2.2. Bifurcation points. An efficient multigrid method for locating singularities or even an envelope of singular points is now described. Basically the problem (2.9)-(2.11) needs to be solved. An FMG algorithm can be designed to solve these equations. The first thing that must be described is the relaxation method.

Relaxation. As before the relaxation will involve local and global steps. The local step is a Newton-Kaczmarz on coarse levels and Newton-Gauss-Seidel on fine ones for the first constraint equation. The next one, which is linear in φ, can be relaxed on coarse levels by Kaczmarz and on fine levels by Gauss-Seidel (see [BT]). Because of the singularity of the linearized part of the system at a bifurcation point, difficulties arise in the coarsening process. The coarse grids do not represent well the singular (or nearly singular) components in u. Two possibilities exist here for correcting the situation. One is to use on the coarse grids a modified multigrid method in the spirit of [BT]. The other possibility is to free u from errors in the singular subspace on the finest level before coarsening takes place. This is similar to the treatment of singular problem as described in [H]. It can be done only by using a global step there since local ones are very slow to converge for these components. The global step includes changing the singular components of u, changing λ, and at the same time minimizing μ^2. For simplicity of exposition we assume that the singular subspace is of dimension one. We use the same global step as the one used for locating limit points, except that now β, δ solve a minimization problem. Namely, the following:

global step 1.

$$u \leftarrow u + \beta\varphi, \qquad \lambda \leftarrow \lambda + \delta \tag{3.13}$$

where (β, δ) solve the following minimization problem:

$$\min_{(\beta,\delta)} \mu^2 \quad \text{subject to} \tag{3.14}$$

$$\langle L(u+\beta\varphi, \lambda+\delta), \varphi\rangle = \langle F_1, \varphi\rangle, \tag{3.15}$$

$$\langle L_u(u+\beta\varphi, \lambda+\delta)\varphi - \mu\langle\varphi, \varphi\rangle = \langle F_2, \varphi\rangle. \tag{3.16}$$

This by itself is not enough since the second constraint equation (2.11) will converge very slowly in the φ direction. This can be taken care of by changing μ on the coarsest levels in the following way:

global step 2.

$$\varphi \leftarrow \gamma\varphi, \qquad \mu = \frac{\langle G - L_u(u, \lambda)\varphi, \varphi\rangle}{\langle\varphi, \varphi\rangle} \tag{3.17}$$

where γ is such that the norm constraint is satisfied. Note that a change of μ in this way is effectively relaxing this equation in the φ direction.

To summarize, the algorithm is basically an FMG algorithm using the FAS formulation where the local relaxation is combined with two global steps. One is performed on the finest level and the other on the coarsest levels. The minimization problem (3.14)-(3.16) is solved by approximating it by a minimization problem obtained by expanding $L(u+\beta\varphi, \lambda+\delta)$ and $L_u(u+\beta\varphi, \lambda+\delta)\varphi$ by Taylor series taking up to quadratic terms in the first one and only linear terms in the second; that is,

$$\min_{(\beta,\delta)} \mu^2 \quad \text{subject to} \tag{3.18}$$

$$\beta A + \delta B + \tfrac{1}{2}\beta^2 C + \beta\delta D + \tfrac{1}{2}\delta^2 E = \langle F_1 - L(u, \lambda), \varphi\rangle, \tag{3.19}$$

$$\beta C + \delta D - \mu\langle\varphi, \varphi\rangle = \langle F_2 - L_u(u, \lambda)\varphi, \varphi\rangle \tag{3.20}$$

where

$$A = \langle L_u(u, \lambda)\varphi, \varphi\rangle, \quad B = \langle L_\lambda(u, \lambda), \varphi\rangle, \quad C = \langle L_{uu}(u, \lambda)\varphi\varphi, \varphi\rangle,$$

$$D = \langle L_{u\lambda}\varphi, \varphi\rangle, \qquad E = \langle L_{\lambda\lambda}, \varphi\rangle.$$

Multigrid Algorithm MGLBP. Consider a sequence of grids $\Omega_k (k \leqq M)$ with mesh sizes h_k satisfying $2h_{k+1} = h_k$. Suppose on each grid operators (L^k, L_u^k) are given in such a way that (L^k, L_u^k) $(k<M)$ is an approximation to (L^{k+1}, L_u^{k+1}) and the Ω_k grid problem is

$$\min_{(u^k,\varphi^k,\lambda)} \mu^2 \quad \text{subject to} \tag{3.21}$$

$$L^k(\tilde{u}^k, \lambda) = F^k, \tag{3.22}$$

$$L_u^k \varphi^k - \mu\varphi^k = G^k, \tag{3.23}$$

$$\|\varphi\|_k^2 = H^k. \tag{3.24}$$

Assume also that interpolation operators I_{k-1}^k, from coarse to fine grids, and restriction operators I_k^{k-1}, $\bar{I}_k^{k-1}$, from fine to coarse grids, are given.

Given an approximate solution (u^k, φ, λ) to the problem (3.21)-(3.24), the multigrid cycle for improving it is denoted by

$$(u^k, \varphi^k, \lambda) \leftarrow \mathrm{MGLBP}(k, u^k, \varphi^k, \lambda, F^k, G^k, H^k) \tag{3.25}$$

and is defined recursively as follows:

```
if $k=1$ then solve (3.21)-(3.24) by enough relaxations (to achieve a desired accuracy)
else
    • Perform $\nu_1$ relaxation sweeps on (3.21)-(3.24), starting with $(u^k, \varphi^k, \lambda)$ and
      resulting in a new approximation $(\bar{u}^k, \bar{\varphi}^k, \bar{\lambda})$.
    • Starting with $u^{k-1} = \bar{I}_k^{k-1}\bar{u}^k$, $\varphi^{k-1} = \bar{I}_k^{k-1}\bar{\varphi}^k$ make $\gamma$ successive cycles of the type
          $(u^{k-1}, \lambda^{k-1}, \lambda) \leftarrow \mathrm{MGLBP}(k-1, u^{k-1}, \varphi^{k-1}, \lambda, F^{k-1}, G^{k-1}, H^{k-1})$
      where
          $F^{k-1} = I_k^{k-1}(F^k - L^k\bar{u}^k) + L^{k-1}\bar{I}_k^{k-1}\bar{u}^k$
          $G^{k-1} = I_k^{k-1}(G^k - L_u^k\bar{\varphi}^k) + L_u^{k-1}\bar{I}_k^{k-1}\bar{\varphi}^k$
          $H^{k-1} = (H^k - \|\bar{\varphi}^k\|_k^2) + \|\bar{I}_k^{k-1}\bar{\varphi}^k\|_{k-1}^2.$
    • Calculate $\bar{u}^k = \bar{u}^k + I_{k-1}^k(u^{k-1} - \bar{I}_k^{k-1}\bar{u}^k)$, $\bar{\varphi}^k = \bar{\varphi}^k + I_{k-1}^k(\varphi^{k-1} - \bar{I}_k^{k-1}\bar{\varphi}^k)$.
    • Perform $\nu_2$ relaxation sweeps on (3.21)-(3.24) starting with $\bar{u}^k$, $\bar{\varphi}$, $\lambda$ and yielding
      $(u^k, \varphi^k\lambda)$, the final result of (3.25).
```

FMGLBP ALGORITHM. To obtain full efficiency, the first approximation on a given level is obtained from a solution of the same problem on the next coarser level, which itself has been calculated in a similar way. The resulting algorithm is called (FMGLBP) and is described next.

Let Π_{k-1}^k be an interpolation operator (usually of higher order than I_{k-1}^k). Given the problem (3.21)-(3.24) with $k=M$, the N-FMG solution of that problem is:

```
    Initial setup.
Set $F^M = 0$, $G^M = 0$, $H^M = 1$.
for $k = M-1, \cdots, 1$ do:
    • $F^k = I_{k+1}^k F^{k+1}$, $G^k = I_{k+1}^k G^{k+1}$, $H^k = H^{k+1}$.
    N-FMG ALGORITHM.
Calculate $u^1$ the solution of (3.31)-(3.34) for $k=1$ by several relaxations.
for $k = 2, \cdots, M$ do:
    • Calculate $u^k \leftarrow \Pi_{k-1}^k u^{k-1}$, $\varphi^k \leftarrow \Pi_{k-1}^k \varphi^{k-1}$.
    • Perform the cycle $(u^k, \varphi^k, \lambda) \leftarrow \mathrm{MGLBP}(k, u^k, \varphi^k, F^k, G^k, H^k)$ $N$ times.
```

4. Numerical experiments. In this section we demonstrate some of the algorithms presented in the paper. We consider partial differential equations of the form

$$\Delta u + f(x, y, u, \lambda) = 0, \qquad (x, y) \in (0, 1) \times (0, 1). \tag{4.1}$$

The discretization used for all cases is the standard five-point Laplacian for the Δu term in the equation, and a pointwise approximation to $f(x, y, u, \lambda)$. The grids used were equally spaced in both directions.

4.1. Locating limit points. To demonstrate the algorithm for locating limit points, we have chosen the well-known Bratu problem. That is,

$$f(x, y, u, \lambda) = \lambda\, e^u. \tag{4.2}$$

A bifurcation diagram for this problem is given schematically in Fig. 1.

Table 1 shows the results of algorithm FMGLLP. The coarsest grid has a mesh size of $h_0 = .25$. Bilinear interpolation used for I^k_{k-1}, bicubic for Π^k_{k-1}, and a nine-point full weighting operator for I^{k-1}_k. The following values were used for the different parameters: $\nu_1 = 1$, $\nu_2 = 2$, and $\gamma = 2$. The local relaxation was employed in lexicographic ordering.

The residuals given in Table 1 are for the currently finest level at the end of the cycle. The two residuals at each row (starting at the left one) are for (3.19), (3.20)

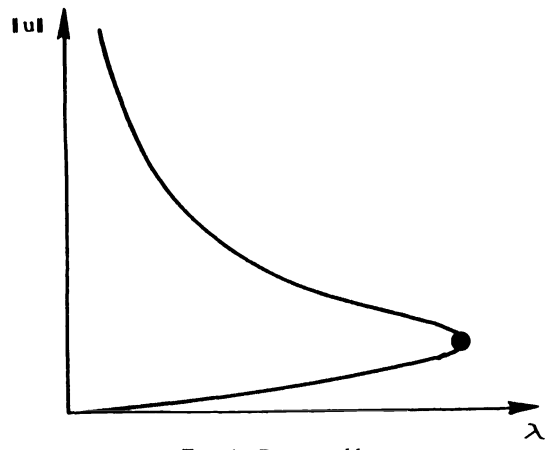


FIG. 1. *Bratu problem.*

TABLE 1
Locating a limit point.

Level	Cycle No.	$\|\text{residuals}\|_2$		λ	$\|u\|$	$\langle L_\lambda, \varphi\rangle$
1	5	0.332E−14	0.435E−14	6.69051	.57217	1.86767
2	1	0.390E−1	0.111E+1	6.79469	.67252	1.98973
	2	0.172E−2	0.123E−1	6.78333	.67258	1.99357
3	1	0.135E−1	0.417E+0	6.80282	.69854	2.02417
	2	0.308E−3	0.136E−1	6.80217	.69857	2.02440
4	1	0.421E−2	0.130E+0	6.80669	.70515	2.03212
	2	0.129E−3	0.213E−2	6.80665	.70515	2.03214
5	1	0.180E−2	0.351E−1	6.80776	.70680	2.03408
	2	0.603E−4	0.209E−3	6.80775	.70680	2.03408

respectively. It is clearly seen that 1-FMG solves for the location of the limit point to the level of discretization errors. The approximation to the limit point λ_*^h obtained by 1-FMG converges as $O(h^2)$. The same rate is obtained for $\langle L_\lambda^h, \varphi^h \rangle$.

4.2. Locating bifurcation points. In this section the algorithm FMGLBP is demonstrated. Two cases are given for which the differential equation has bifurcations at $\lambda = \pi^2(m^2+n^2)$ where n, m are integers. A simple way of constructing a problem in the form of (4.1) for which bifurcations exist at the above mentioned λ's (see [K]) is to begin by prescribing a branch of solutions of the form $u(x, y, \lambda) = q(\lambda)\bar{U}_0(x, y)$ where $q(\lambda)$, $\bar{U}_0(x, y)$ are given. This is achieved if

$$f(x, y, u(x, y, \lambda), \lambda) = -\Delta u(x, y, \lambda).$$

For branches to bifurcate from this branch at the above λ's, we require that

$$f_u(x, y, u(x, y, \lambda), \lambda) = 1.$$

As a special case of this we take

$$f(x, y, u, \lambda) = -q(\lambda)\Delta \bar{U}_0(x, y) + \lambda P(u - q(\lambda)\bar{U}_0(x, y)), \qquad P(z) = z + z^2. \tag{4.3}$$

We consider two cases: (i) $\bar{U}_0(x, y) = x(x-1)y(y-1)$, and (ii) $\bar{U}_0(x, y) = \sin(\pi x)\sin(\pi y)$. The first is a case of a perfect bifurcation for the discrete levels while the other is an imperfect bifurcation for the discrete levels. Tables 2 and 3 show the result of the FMGLBP Algorithm for locating the first bifurcation point. Intergrid transfers used are identical to the ones used in § 4.1. The two residuals (left and right)

TABLE 2
Locating a bifurcation point: Perfect bifurcation.

Level	Cycle No.	$\lVert\text{residuals}\rVert_2$		λ	$\lVert u \rVert$	$\langle L_\lambda, \varphi \rangle$
1	5	0.915E−10	0.176E−7	18.74516	.1226	−6.58175E−8
2	1	0.264E−2	0.381E+0	19.28335	.14578	3.64700E−3
	2	0.725E−3	0.387E−1	19.46164	.14329	4.43231E−4
3	1	0.509E−3	0.264E−1	19.67002	.14831	1.01890E−4
	2	0.364E−4	0.336E−1	19.67527	.14721	1.04083E−5
4	1	0.715E−4	0.280E−2	19.72318	.14829	2.98171E−6
	2	0.235E−5	0.110E−3	19.72336	.14793	−3.31378E−8
5	1	0.133E−4	0.297E−3	19.73545	.14817	−3.63521E−6
	2	0.974E−7	0.129E−3	19.73528	.14806	−7.34759E−7

TABLE 3
Locating a bifurcation point: Imperfect bifurcation.

Level	Cycle No.	$\lVert\text{residuals}\rVert_2$		λ	$\lVert u \rVert$	$\langle L_\lambda, \varphi \rangle$
1	5	0.186E−13	0.628E−13	9.48662	1.5945	0.56972
2	1	0.160E−1	0.545E+0	13.46777	1.8857	0.20472
	2	0.291E−4	0.125E−1	13.46833	1.9076	0.20487
3	1	0.308E−1	0.901E+0	16.21966	2.0428	0.85237E−1
	2	0.149E−2	0.400E−1	16.21687	2.0632	0.85390E−1
4	1	0.510E−2	0.591E+0	17.86465	2.1293	0.38793E−1
	2	0.113E−3	0.135E−1	17.86442	2.1419	0.38813E−1
5	1	0.829E−3	0.335E+0	18.77085	2.1746	0.18483E−1
	2	0.542E−5	0.421E−2	18.77096	2.1815	0.18484E−1

are for the first and second constraint equations (3.32), (3.33), respectively. The approximated singular point λ_*^h converges at a rate that looks like $O(h)$ for the imperfect bifurcation case and like $O(h^2)$ for the other case. Also note that $\langle L_\lambda^h, \varphi^h \rangle$ in case (ii) converges to zero as $O(h)$. As the actual value of this quantity shows, it is not enough to consider single grid experiments when locating bifurcation points. A refinement is essential for distinguishing between actual limit points and imperfect discrete bifurcation. The results of Tables 2 and 3 clearly demonstrate the effectiveness of the algorithm described.

REFERENCES

[B] A. BRANDT, *Multigrid Techniques*: 1984 *Guide, with Applications to Fluid Dynamics.* Available as GMD studien Nr. 85, GMD-AIW, Postfach 1240, D-5205, St. Augustin 1, FRG, 1984.

[BK] J. BOLSTAD AND H. B. KELLER, *A multigrid continuation method for elliptic problems with folds*, SIAM J. Sci. and Statist. Comput., 7 (1986), pp. 1081-1104.

[BMR] A. BRANDT, S. MCCORMICK, AND J. RUGE, *Multigrid algorithms for differential eigenproblems*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 244-260.

[BRP] F. BREZZI, J. RAPPAZ, AND P. A. RAVIART, *Finite dimensional approximation of nonlinear problems*, Numer. Math., 38 (1981), pp. 1-30.

[BT] A. BRANDT AND S. TA'ASAN, *Multigrid method for slightly indefinite and nearly singular problems*, in Multigrid Methods II, Lecture Notes in Mathematics 1228, Springer-Verlag, Berlin, New York, 1985.

[DK] D. W. DECKER AND H. B. KELLER, *Path following near bifurcation*, Comm. Pure Appl. Math., 34 (1981), pp. 149-175.

[H] W. HACKBUSCH, *Multi-Grid Methods and Applications*, Springer-Verlag, Berlin, New York, 1985.

[HT] W. HACKBUSCH AND U. TROTTENBERG, EDS., *Multigrid methods*, Lecture Notes in Mathematics 960, Springer-Verlag, Berlin, New York, 1982, pp. 1-176.

[K] H. B. KELLER, *Numerical solutions of bifurcation and nonlinear eigenvalue problems*, in Applications of Bifurcation Theory, P. Rabinowitz, ed., Academic Press, New York, 1977, pp. 359-384.

[KK] J. P. KEENER AND H. B. KELLER, *Perturbed bifurcation theory*, Arch. Rational Mech. Anal., 50 (1973), pp. 159-175.

[M] H. D. MITTLEMANN, *Multigrid methods for simple bifurcation problems*, in Lecture Notes in Mathematics 960, Springer-Verlag, Berlin, New York, 1982, pp. 558-575.

[Moo] G. MOORE, *The numerical treatment of non trivial bifurcation points*, Numer. Funct. Anal. Optim., 2 (1980), pp. 441-472.

[MS] G. MOORE AND A. SPENCE, *The calculation of turning points of nonlinear equations*, SIAM J. Numer. Anal., 17 (1980), pp. 567-576.

[R] W. RHEINBOLDT, *Computation of critical boundaries on equilibrium manifolds*, SIAM J. Numer. Anal., 19 (1982), pp. 653-669.

[ST] K. STUBEN AND U. TROTTENBERG, *Multigrid methods: Fundamental algorithms, model problem analysis and applications*, in Lecture Notes in Mathematics 960, Springer-Verlag, Berlin, New York, 1982.

[SW] A. SPENCE AND B. WERNER, *Non-simple turning points and cusps*, IMA J. Numer. Anal., 2 (1982), pp. 413-427.

[T] S. TA'ASAN, *Multigrid methods for bifurcation problems, The self adjoint case*, ICASE Report No. 87-40, ICASE, NASA Langley Research Center, Hampton, VA, 1987.

SIAM J. Sci. Stat. Comput.
Vol. 11, No. 1, pp. 63-81, January 1990

A NUMERICAL ALGORITHM FOR STABILITY ANALYSIS OF DIFFERENCE METHODS FOR HYPERBOLIC SYSTEMS*

MICHAEL THUNÉ†

Abstract. The stability theory for difference approximations of hyperbolic initial boundary value problems is based on normal mode analysis. To perform such a stability investigation analytically is very difficult even for low-order approximations of scalar problems. For more complicated cases some numerical technique must be used. Here, a numerical algorithm designed for this purpose is presented. It can handle one- and two-dimensional problems and can easily be extended to higher-dimensional cases. The algorithm is justified by a theoretical analysis and experiments show that it is reliable and efficient.

Key words. stability, initial boundary value problems, hyperbolic systems, difference approximations

AMS(MOS) subject classification. 65M10

1. Introduction. Stability is a key concept in the numerical solution of time-dependent partial differential equations (pde). Here, we will consider the stability of difference approximations of first-order hyperbolic initial boundary value problems. We treat systems in two space dimensions, but the results can be extended easily to higher-dimensional systems.

The question we address is how can stability be analyzed in *practice*? The stability theory is well known [7], [9]. It is based on normal mode (NM) analysis. However, to actually perform such an analysis is very difficult even for small problems and low-order accurate approximations. Most of the cases reported in the literature concern problems that are small enough to be treated analytically, mostly with considerable effort. The authors that have treated more difficult cases (e.g., [11], [14]) have used numerical techniques that do not take advantage of the special properties of the NM analysis problem.

Some years ago, we reported on a first version of the software system Ibstab [16]. It included a numerical algorithm specially tailored for the NM analysis of one-dimensional problems. In the present paper, we present a new version of the algorithm, for two-dimensional problems. The outline of the paper is as follows. In § 2, we describe earlier, related research. In §§ 3-4, the new algorithm is presented and analyzed. Some additional comments are given in § 5 and finally, in § 6, some test results are discussed.

The algorithm presented here is available in a Fortran implementation, which can be ordered from the author on a noncommercial basis.

2. A survey of earlier related research.

2.1. The stability theory. Our model problem will be the linear, hyperbolic initial boundary value problem

$$u_t = A_1 u_x + A_2 u_y, \qquad 0 \leqq x < \infty, \quad -\infty < y < \infty, \quad 0 \leqq t, \tag{1}$$

$$u(x, y, 0) = 0, \tag{2}$$

$$Bu(0, y, t) = f(y, t), \tag{3}$$

$$u(x) \in \mathscr{L}_2(0, \infty). \tag{4}$$

* Received by the editors May 25, 1988; accepted for publication (in revised form) November 30, 1988. This work was supported by the Swedish Natural Science Research Council.

† Department of Scientific Computing, Uppsala University, Sturegatan 4B—2nd floor, S-75223 Uppsala, Sweden.

We take u to be a $d \times 1$-vector. A_1 and A_2 are constant $d \times d$-matrices and B is a matrix that makes (1)-(4) well posed. For problems with two boundaries in the x-direction and lower-order terms, the stability investigation can be reduced to the analysis of (1)-(4). Under additional assumptions the same is true for problems with variable coefficients and inhomogeneous initial data.

We consider some difference approximation of (1)-(4), with solution $u_{jk}^n \approx u(x_j, y_k, t_n)$, where (x_j, y_k, t_n) is a point in the computational grid. The mesh sizes in the grid are Δx, Δy, and Δt. For simplicity, we assume $\Delta y = \Delta x$. The crucial parameter for the stability of the approximation is $\lambda \equiv \Delta t / \Delta x$.[1]

The theory of Michelson [9] gives necessary and sufficient stability conditions assuming a domain as in (1), i.e., where all except one of the space directions are unbounded, and assuming that the difference approximation is dissipative. For more general domains and nondissipative approximations the conditions are necessary.

The NM stability analysis starts out with the ansatz

$$u_{jk}^n = z^n e^{i\xi k} v_j \tag{5}$$

where z is a complex scalar, $\xi \equiv \omega \Delta y$ comes from a Fourier transform in the y direction, and v_j is a $d \times 1$-vector. By inserting (5) into the approximation of (1), we obtain *the resolvent equation*, a system of ordinary difference equations for the components of v_j. The related *characteristic equation* is

$$P(z, \kappa, \xi, \lambda) = 0 \tag{6}$$

where P is a polynomial in z, κ, λ and a trigonometric polynomial in ξ.

Let us now freeze z, ξ, and λ at arbitrary values, such that $|z| > 1$ and $\lambda < \lambda_{cp}$, where λ_{cp} is the stability limit for the Cauchy problem associated with (1)-(4). The condition on λ implies that (6) will have no solutions with $|z| > 1$, $|\kappa| = 1$. Thus, for our frozen z, ξ, λ, the solutions of (6) will split into two disjoint sets $M_<(z, \xi, \lambda)$ and $M_>(z, \xi, \lambda)$:

$$M_<(z, \xi, \lambda) = \{\kappa(z, \xi, \lambda); |\kappa| < 1\}, \qquad M_>(z, \xi, \lambda) = \{\kappa(z, \xi, \lambda); |\kappa| > 1\}.$$

Now, let κ_ν, $\nu = 1, \cdots, N$, be the elements of $M_<(z, \xi, \lambda)$, counted with algebraic multiplicity. The general solution $v_j \in l_2(0, \infty)$ of the resolvent equation has the form

$$v_j = \sum_{\nu=1}^{N} \sigma_\nu \Phi_\nu(j) \kappa_\nu^j \tag{7}$$

where $\Phi_\nu(j)$ are vectors and σ_ν are arbitrary scalar coefficients.

The values of the coefficients σ_ν are determined by inserting (5) and (7) into the numerical boundary conditions, which can be expressed in the general form:

$$\sum_{\tau=-1}^{s} \sum_{k=-k_1}^{k_2} \sum_{j=0}^{q} B_{\mu jk}^{(\tau)} u_{jk}^{n-\tau} = f_\mu, \qquad \mu = 0, \cdots, l-1.$$

Here, $B_{\mu jk}^{(\tau)}$ are constant $d \times d$-matrices. The equations for σ_ν, $\nu = 1, \cdots, N$, then are

$$\sum_{\nu=1}^{N} \left[\sum_{\tau=-1}^{s} z^{n-\tau} \sum_{k=-k_1}^{k_2} e^{i\xi k} \sum_{j=0}^{q} B_{\mu jk}^{(\tau)} \Phi_\nu(j) \kappa_\nu^j \right] \sigma_\nu = f_\mu, \qquad \mu = 0, \cdots, l-1. \tag{8}$$

[1] A more general treatment is achieved by assuming $\Delta y = \varphi \Delta x$. The stability analysis can then be performed in terms of λ for different choices of φ.

This is a linear system of equations, the coefficient matrix of which we denote by $G(z, \kappa_1, \cdots, \kappa_N, \xi, \lambda)$. We further introduce the notation $g(z, \kappa_1, \cdots, \kappa_N, \xi, \lambda) \equiv \det(G(z, \kappa_1, \cdots, \kappa_N, \xi, \lambda))$. The condition for a nontrivial solution to (8) is

$$g(z, \kappa_1, \cdots, \kappa_N, \xi, \lambda) = 0, \tag{9}$$

which will be referred to as *the determinant condition.*

The equations (6) and (9) form an *implicit stability condition* on λ: the approximation of (1)-(4) is *un*stable for those values of λ for which (6), (9) has solutions with $|z| > 1$, $|\kappa_\nu| < 1$, $\nu = 1, \cdots, N$. In addition, we have to pay special attention to all solutions with $|z| = 1$. For further details, the reader is referred to [7] and [9].

Performing an NM stability analysis thus means deriving and solving (6) and (9). For each fixed λ we have to solve these equations for all $\xi \in [0, 2\pi)$. Our aim is to find the smallest λ for which the approximation is unstable.

There are two different ways of looking at (6), (9). The first and most natural one will subsequently be called *the scalar view.* For fixed λ and ξ our problem can be regarded as being that of solving (9), the unknowns $\kappa_1(z), \cdots, \kappa_N(z)$ being *defined* by (6). When we adopt this view, the determinant condition can be written as $g(z) = 0$, i.e., as a scalar equation in one unknown. The parameters λ and ξ will be included in the parameter list only when they are of importance to the context.

The second way of looking at (6), (9) will be referred to as *the systems view.* This is to see (6), (9) as a system of equations for $z, \kappa_1, \cdots, \kappa_N$, the system being parametrized by ξ and λ. This view is natural in some one-dimensional cases, for the following reason. By definition, the coefficient matrix of a linear, hyperbolic, first-order system in one space dimension can be diagonalized. Thus, a suitable transformation of variables will take the pde system into a set of d uncoupled, scalar equations. For many difference methods this implies that (6) will factor into d factors, one for each pde:

$$P(z, \kappa, \xi, \lambda) = \prod_{i=1}^{d} P_i(z, \kappa, \xi, \lambda).$$

We thus get d characteristic equations

$$P_i(z, \kappa^{(i)}, \xi, \lambda) = 0, \qquad i = 1, \cdots, d \tag{10}$$

where equation i will contribute μ_i elements to the set $M_<(z, \xi, \lambda)$. In the case of a centered, second-order difference approximation in space, we will have $\mu_i = 1$, $i = 1, \cdots, d$. Consequently, $N = d$ and (10), (9) will be a set of $N+1$ equations for the $N+1$ unknowns $z, \kappa_1, \cdots, \kappa_N$. When we analyze higher-order approximations, then $\mu_i > 1$ and we have more unknowns than equations. However, we can still look on (10), (9) as being a system of $N+1$ equations in $N+1$ unknowns by letting P_i be repeated μ_i times.

2.2. The analysis in practice. The essential task when performing the stability analysis is to solve (6), (9) in order to find any solution with $|z| \geqq 1$, $|\kappa_\nu| \leqq 1$, $\nu = 1, \cdots, N$. For each fixed λ we must consider all $\xi \in [0, 2\pi)$. The analyses that have been reported in the literature, with few exceptions, concern one-dimensional problems (i.e., the special case $\xi \equiv 0$). The common approach has been to adopt the systems view of (6), (9). Oliger [11] used a reduction technique, by which he reduced the system into *one* polynomial in one unknown. This polynomial, of a very high degree, was then solved by a standard numerical method. This was a very time consuming way of solving (6), (9). Another way of using the systems view (e.g., [4], [14]) has been to solve (10), (9) by means of some general continuation method, for fixed, predetermined values of λ. The primary drawback of this approach is that it means solving for *all* solutions of the system, including those with $|z| \ll 1$. This is inefficient, as we are only interested in the case $|z| \geqq 1$.

A conceptually different approach has been suggested by Goldberg and Tadmor. They have treated a fairly general class of difference approximations of problems in one space dimension. By exploring the properties of this class they have been able to simplify (6), (9), thus arriving at what they call *convenient* stability conditions for these approximations [5]. The Goldberg/Tadmor conditions simplify the stability analysis considerably. However, they are limited in scope, in that

- The class to which they apply does not include all the approximations worth considering;
- It is crucial for the simplification that the coefficient matrix of the pde system can be diagonalized, which means that the conditions cannot be generalized to multidimensional systems.

2.3. The Ibstab approach. For problems to which the convenient stability conditions do not apply, we must use a numerical technique to perform the stability analysis. To avoid the inefficiencies involved in applying a general method to solve (6), (9), there is need for a special purpose numerical algorithm. In [16], Thuné presented such an algorithm for the one-dimensional case. It was embedded in a software system called Ibstab (from "Initial boundary value problem stability analysis"). This algorithm starts out with the scalar view, searching for solutions in the set

$$\{(z, \lambda); \lambda > 0, |z| = 1 + \eta\}$$

where $\eta > 0$ is a small parameter. The essential improvement here is that we only need to look for solutions on *the search circle* $|z| = 1 + \eta$. This is possible, as is shown in § 4, due to properties of (6), (9) and of the underlying difference approximation. Presently, it is sufficient to note that the premises of the algorithm are

- For $\lambda = 0$ all solutions $z(\lambda)$ are on or inside the unit circle in the complex plane;
- A slight increase in λ will only cause a slight change in the solutions $z(\lambda)$; no new solutions will enter.

Whenever the algorithm finds some $z = z_0$, that is close to being a solution, it switches to the systems view and starts an iterative procedure for solving (10), (9) with $(z_0, \kappa_1(z_0), \cdots, \kappa_N(z_0))$ as initial guess.

The basic structure of the Ibstab numerical algorithm is shown below. The following notation is used:

next-lambda—a procedure for computing a new, increased λ value,
next-theta—a procedure for computing a new increased value of $\theta \equiv \arg(z)$,
close(z, λ)—a logical procedure that is true if z is close to being a solution to $g(z) = 0$ for a fixed λ,
solve(α)—a procedure for solving (10), (9) iteratively, using the point α as initial guess,
$(z^*, \kappa_1^*, \cdots, \kappa_N^*)$—a solution of (10), (9), obtained by *solve.*

The exact definitions of *next-lambda, next-theta, close,* and *solve* are not important to the basic idea of the algorithm, but will be discussed in detail in § 3, when we consider the efficient implementation of the algorithm.

ALGORITHM Basic Ibstab.

```
for $\lambda = \lambda_1$ ($\lambda \leftarrow$ *next-lambda*) $\lambda_s$ do
    for $\theta = 0$ ($\theta \leftarrow$ *next-theta*) $2\pi$ do[2]
        $z \leftarrow (1+\eta)\, e^{i\theta}$
        $(\kappa_1, \cdots, \kappa_N) \leftarrow M_<(z, \lambda)$
        if *close*$(z, \lambda)$ then
            *solve*$(z, \kappa_1, \cdots, \kappa_N)$
            if *convergence* then
                *check-stability-criteria*$(z^*, \kappa_1^*, \cdots, \kappa_N^*)$
            endif
        endif
    endfor
endfor
```

The algorithm terminates when we find a solution $(z^*, \kappa_1^*, \cdots, \kappa_N^*)$ which, according to the stability criteria, shows that the difference approximation is unstable, or, if no such solution is found, when $\lambda > \lambda_s$. The value of the parameter λ_s is given by the user of the algorithm. If λ_{cp} is known, then $\lambda_s = \lambda_{cp}$ is the natural choice.

The results presented in [16] for an implementation of this algorithm show that the Ibstab approach is efficient.

3. The new algorithm. The development of the Ibstab algorithm since the publication of [16] has followed three directions: a more efficient implementation of details, a better theoretical justification, and an extension to problems in two space dimensions. The theoretical justification is treated in § 4. Here, we present the new two-dimensional version of the algorithm, including the improved implementation details.

There are two difficulties involved in going from one to two space dimensions. The first is that in the two-dimensional case we cannot assume that the pde system can be diagonalized. Thus, the formulation (10) of the characteristic equations is no longer possible, which makes it less natural to adopt the systems view of (6), (9). The other difficulty is that the search space will now include the parameter ξ, coming from the Fourier transformation in the y direction.

There are two possible ways to resolve the first difficulty. The systems view is adopted only in *solve* and thus one way to avoid the difficulty would be to adopt the scalar view in *solve* as well. However, we can look on (6), (9) as being a system of $N+1$ equations in $N+1$ unknowns by repeating (6) N times, which is the second possibility. To adopt the scalar view in *solve* would mean that *solve* should be a procedure for solving $g(z)=0$, where $g(z) \equiv g(z, \kappa_1(z), \cdots, \kappa_N(z))$. Here, $\kappa_j(z)$, $j = 1, \cdots, N$, are defined by (6). This approach could have been used already in the one-dimensional case and the reason for not using it is the following. In cases when $|d\kappa/dz|$ is large, the procedure of solving for $\kappa_j(z)$ from (6) is not sufficiently well conditioned to allow for convergence with the scalar approach. Experiments with *solve* based on Newton-Raphson's method confirm this. Another fact that makes it troublesome to base *solve* on the scalar view is that the set $M_<(z, \xi, \lambda)$ is not defined for $|z| \leqq 1$. This means that if *solve* generates an iterate z with $|z| \leqq 1$, then it is difficult to know which of the solutions κ of (6) should be chosen for this z.

The second possibility, to use the systems view in the two-dimensional case by repeating (6) N times, seems liable to be ill-conditioned as well, for the following

[2] As pointed out by one of the referees, it is in the common case with real coefficients in the pde system enough, by symmetry, to consider $\theta \in [0, \pi]$.

reasons. First, with $N \geqq 2$ the systems view may lead to false solutions if *solve* is called far from a true solution. Two components of the initial guess, $\kappa_i(z_0)$ and $\kappa_j(z_0)$, corresponding to the same characteristic equation, may then converge to the same value κ^*. Second, in the situations where a scalar *solve* failed, the systems *solve* might be expected to come into difficulties as well. However, the first is no problem, because we assume *solve* to be called very close to a solution.[3] The second difficulty can be dealt with successfully by using a solver that is robust in treating systems with a nonsingular or ill-conditioned Jacobian matrix. Among the test cases reported in § 6 are several where one or more characteristic equations had to be repeated. With *solve* based on Powell's hybrid method these cases caused no problems. The conclusion from this discussion is that it is preferable to base *solve* on the systems view.

We now turn to the problem of how to treat the new variable ξ. Here, too, there are two possibilities. One would be to treat ξ in the same spirit as we treat λ and θ in the basic one-dimensional algorithm. This would mean including a loop over ξ from zero to 2π, with a possibly variable stepsize $\delta\xi$. The second possibility would be to find *all* solutions for $\xi = 0$, say, and then trace these solutions by continuation in ξ. However, for efficiency reasons we have already rejected the idea of finding all solutions, since these include those that are of no interest in the stability analysis.

Consequently, we have chosen to treat ξ in analogy with the treatment of λ and θ. We derive a variable stepsize $\delta\xi$ in the following way. Let $z(\xi)$ denote a solution path for a fixed $\lambda = \lambda_0$. It is defined by

$$g(z(\xi), \kappa_1(z(\xi)), \cdots, \kappa_N(z(\xi)), \xi, \lambda_0) \equiv 0, \tag{11}$$

i.e., the determinant condition is satisfied identically. We can write $z(\xi)$ in polar coordinates as

$$z(\xi) = \zeta(\xi)\, e^{i\theta(\xi)}, \qquad \zeta(\xi) \in R_+, \quad \theta(\xi) \in R. \tag{12}$$

Consider a fixed $\xi = \xi_0$ and $z(\xi_0)$ such that $\zeta(\xi_0) < 1$. We need to choose $\delta\xi > 0$, in such a way that

$$1 \geqq \zeta(\xi_0 + \delta\xi) \approx \zeta(\xi_0) + \frac{d\zeta(\xi_0)}{d\xi}\,\delta\xi.$$

There are two cases:

(i) $d\zeta(\xi_0)/d\xi \leqq 0$, implies no restriction on $\delta\xi$;

(ii) $d\zeta(\xi_0)/d\xi > 0$, implies that we can at most allow $\delta\xi = (1 - \zeta(\xi_0))/(d\zeta(\xi_0)/d\xi)$. Considering that the argument above was based on a Taylor expansion, and is thus valid only for sufficiently small $\delta\xi$, we must impose an upper limit $\bar{\delta\xi}$. Furthermore, adopting the scalar view, let $\tilde{Z}(\xi_0, \lambda_0)$ denote the set of solutions $z(\xi_0, \lambda_0)$ that the algorithm found for $\xi = \xi_0$, $\lambda = \lambda_0$. The formula for $\delta\xi$ can then be expressed as

$$\delta\xi = \min\left[\bar{\delta\xi};\ \min_{\tilde{Z}(\xi_0,\lambda_0)} (1 - \zeta(\xi_0)) \Big/ \frac{d\zeta(\xi_0)}{d\xi}\right]. \tag{13}$$

Of course, we could think of other ways of computing $\delta\xi$, but the one presented here has the advantage of being simple and computationally efficient. To compute $d\zeta(\xi_0)/d\xi$

[3] The only special treatment that is needed is when the procedure *close* gives a false alarm. Then the systems view may lead to false solutions. Such a case is easily detected by checking the multiplicity of κ^*. If the multiplicity is one, then the solution is false. As these false solutions occur only when *solve* is called *far* from a solution, they are harmless.

we use (12), which implies

$$\frac{d\zeta}{d\xi} = \zeta \operatorname{Re}\left(\frac{1}{z}\frac{dz}{d\xi}\right).$$

Here, $dz/d\xi$ can be derived from (11) and (6) by implicit differentiation. This completes the derivation of a stepsize $\delta\xi$.

Before presenting the new algorithm as a whole, we will also comment on the improvements that have been made to the implementation details. In the basic algorithm, the procedures *next-lambda*, *next-theta*, *close*, and *solve* were not specified. It is the precise definitions of these that have been changed to improve efficiency.

The most important change is that of *close*. In the definition we adopt the scalar view, i.e., for a fixed $\lambda = \lambda_0$, $\xi = \xi_0$, we solve $g(z) = 0$. In the early implementation of Ibstab [16], we used

$$close(z, \lambda) \equiv |g(z)| \leqq \varepsilon \vee |g'(z)| \geqq \frac{\varepsilon}{\mu},$$

where ε and μ were small parameters. Numerical experiments show that this leads to unnecessarily many calls to the procedure *solve*, which is the most time consuming part of the algorithm. The definition has now been changed into

$$close(z, \xi, \lambda) \equiv |g(z)|/|g'(z)| \leqq \mu.$$

This has decreased the number of calls to *solve* by a factor of about 5-10, a significant improvement.

The procedure *next-theta* was earlier designed to compute a variable stepsize. The new theoretical analysis (cf. § 4) shows that it is sufficient to use a constant stepsize $\delta\theta$. The procedure *next-lambda* involved some unnecessarily expensive computations [16, p. 966]. However, we may note that the stepsize $\delta\lambda$ can be derived in a way analogous to the derivation of $\delta\xi$. We introduce the following notation:

$\tilde{Z}(\lambda_0)$—the set of solutions $z(\lambda_0)$ that the algorithm found for $\lambda = \lambda_0$,

Δ—we require that for $\lambda = \lambda_0 + \delta\lambda$ all solutions fulfill $|z| \leqq 1 + \Delta$,

$\bar{\delta\lambda}$—an imposed upper limit on the stepsize.

The formula for $\delta\lambda$ will then be

$$\delta\lambda = \min\left[\bar{\delta\lambda};\ \min_{\tilde{Z}(\lambda_0)} (1 + \Delta - \zeta(\lambda_0)) \Big/ \frac{d\zeta(\lambda_0)}{d\lambda}\right]. \tag{14}$$

The changes in *next-theta* and *next-lambda* reduce the amount of computations needed in the algorithm.

Finally, *solve* has been changed. Earlier, Brown's method [3] was used, but it was not sufficiently robust. Presently, Powell's hybrid method is used [10] and it has performed in a very reliable way.

We conclude this section by showing the new algorithm for two space dimensions and with the improved details included.

ALGORITHM Ibstab2.
```
for $\lambda = \lambda_1$ ($\lambda \leftarrow$ next-lambda) $\lambda_s$ do
    $\delta\lambda \leftarrow \bar{\delta\lambda}$
    for $\xi = 0$ ($\xi \leftarrow$ next-xi) $2\pi$ do
        $\delta\xi \leftarrow \bar{\delta\xi}$
        for $\theta = 0$ ($\theta \leftarrow$ next-theta) $2\pi$ do
            $z \leftarrow (1+\eta)\, e^{i\theta}$
```

```
                    $(\kappa_1, \cdots, \kappa_N) \leftarrow M_<(z, \xi, \lambda)$
                    **if** $close(z, \xi, \lambda)$ **then**
                        $solve(z, \kappa_1, \cdots, \kappa_N)$
                        **if** *convergence* **then**
                            $check\text{-}stability\text{-}criteria(z^*, \kappa_1^*, \cdots, \kappa_N^*)$
                            $update\text{-}stepsizes(\delta\xi, \delta\lambda)$
                        **endif**
                    **endif**
                **endfor**
            **endfor**
        **endfor**
        $update\text{-}stepsizes(\delta\xi, \delta\lambda) \equiv$
            **if** $d\zeta^*/d\xi > 0$ **then** $\delta\xi \leftarrow \min[\delta\xi; (1-\zeta^*)/(d\zeta^*/d\xi)]$
            **if** $d\zeta^*/d\lambda > 0$ **then** $\delta\lambda \leftarrow \min[\delta\lambda; (1+\Delta-\zeta^*)/(d\zeta^*/d\lambda)]$
        $next\text{-}lambda \equiv \lambda + \delta\lambda$
        $next\text{-}xi \equiv \xi + \delta\xi$
        $next\text{-}theta \equiv \theta + \delta\theta$
        $close(z, \xi, \lambda) \equiv |g(z)|/|g'(z)| \leqq \mu$
        $solve \equiv$ Powell's hybrid method
```

4. Theoretical justification of Ibstab2.

4.1. The theory. The theoretical arguments are based on the scalar view of (6), (9), i.e., that our problem is to solve $g(z)=0$ for all solutions that fulfill $|z| \geqq 1$. The following notation will be used:

$D(\lambda)$—The difference approximation at interior points of the computational domain,

$n(\lambda)$—The highest time level involved in $D(\lambda)$,

$Z(\lambda)$—$\{z;\ g(z, \lambda, \xi)=0 \text{ for some } \xi\}$.

We begin by making some assumptions and a definition.

Assumption 1. *$D(0)$ is a stable approximation of $u_t = 0$.*

Assumption 2. *$n(\lambda) \leqq n(0)$ for all $\lambda > 0$.*

DEFINITION 1. Consider two distinct λ values, λ_i and λ_j. If, as λ_i changes continuously into λ_j, $Z(\lambda_i)$ changes continuously into $Z(\lambda_j)$, then $Z(\lambda_j)$ is a *perturbation* of $Z(\lambda_i)$.

The following lemmas show that under the above assumptions, the premises of the basic algorithm are fulfilled.

LEMMA 1. *Assumption* 1 *implies that $|z| \leqq 1$ for all $z \in Z(0)$.*

LEMMA 2. *Assumption* 2 *implies that $Z(\lambda)$ is a perturbation of $Z(0)$ for all $\lambda > 0$.*

Lemma 1 is evident. For a proof of Lemma 2, in the one-dimensional case, see [15, § III.2.2].

Next, we restrict the domain of critical solutions, by making the following assumption.

Assumption 3. *The pseudocontinuous variation of λ is made in such a way that for every λ generated by Algorithm* Ibstab2, *we have*

$$z \in Z(\lambda) \Rightarrow |z| \leqq 1 + \Delta.$$

(Formula (14) was designed to fulfill this assumption.) Thus, it is sufficient that Ibstab2 finds all solutions with $1 \leqq |z| \leqq 1 + \Delta$.

Let us finally make an assumption concerning the convergence properties of the procedure *solve.*

Assumption 4. *Let z_0 be a point on the search circle. If, for fixed λ and ξ, the Newton-Raphson method with z_0 as initial guess would converge to a solution z^* of $g(z)=0$, then solve $(z_0, \kappa_1(z_0), \cdots, \kappa_N(z_0)$ converges to $(z^*, \kappa_1(z^*), \cdots, \kappa_N(z^*))$.*

This assumption is very reasonable. The method currently used in *solve*, Powell's hybrid method, is a combination of Newton's method and the gradient method, with the purpose of increasing the robustness compared to the pure Newton's method.

In the remainder of this section, we will show convergence supposing that *solve* is based on solving $g(z)=0$ with Newton-Raphson's method. By Assumption 4, the results will then also hold, for example, to Powell's hybrid method. We let z^* denote a solution of $g(z)=0$. Furthermore, we introduce $C(\beta)\equiv\{z; |z-z^*|\leqq\beta\}$. First, we state a general lemma.

LEMMA 3. *Assume that $g'(z^*)\neq 0$ and that*

$$z_1, z_2 \in C(\beta) \Rightarrow \left|\frac{g''(z_1)}{g'(z_2)}\right| < \frac{\sqrt{3}-1}{\beta}.$$

Then the Newton-Raphson method applied to $g(z)=0$ will converge to z^* for all initial guesses $z_0 \in C(\beta)$.

Proof. The lemma is proved by showing that the assumptions imply that

$$z \in C(\beta) \Rightarrow \left|\frac{g(z)g''(z)}{[g'(z)]^2}\right| < 1.$$

We have

$$0 = g(z^*) = g(z) + g'(z)(z^*-z) + \tfrac{1}{2}g''(z_I)(z^*-z)^2$$

for some intermediate point z_I. This implies

$$\left|\frac{g(z)g''(z)}{[g'(z)]^2}\right| \leqq |z^*-z|\left[1+\frac{1}{2}\left|\frac{g''(z_I)}{g'(z)}\right||z^*-z|\right]\left|\frac{g''(z)}{g'(z)}\right|.$$

For $z \in C(\beta)$, we then we have

$$\left|\frac{g(z)g''(z)}{[g'(z)]^2}\right| < \beta\left[1+\frac{1}{2}\frac{\sqrt{3}-1}{\beta}\beta\right]\frac{\sqrt{3}-1}{\beta} = 1,$$

and the lemma follows. □

Next, we study conditions such that the procedure *solve* will be called.

LEMMA 4. *Assume that $g'(z^*)\neq 0$, $\mu > \beta$, and*

$$z_1, z_2 \in C(\beta) \Rightarrow \left|\frac{g''(z_1)}{g'(z_2)}\right| < \left(\frac{\mu}{\beta}-1\right)\frac{2}{\beta}.$$

Then, the procedure solve will be called for every search point $z_0 \in C(\beta)$.

Proof. By Taylor expansion, we find that the assumptions imply

$$z_0 \in C(\beta) \Rightarrow \left|\frac{g(z_0)}{g'(z_0)}\right| \leqq \mu.$$

Thus, *close*(z_0, ξ_0, λ_0) (where ξ_0 and λ_0 are the current, fixed values of ξ and λ) is true and *solve* is called. □

Finally, we can state Theorem 1.

THEOREM 1. *Assume that the parameters η and μ are chosen such that*

$$\eta \leqq \frac{1}{2}\Delta,$$

$$\mu > \beta = \left[(1+\Delta)^2 - 2(1+\Delta)(1+\eta)\cos\frac{\delta\theta}{2} + (1+\eta)^2\right]^{1/2}.$$

Furthermore, assume that for every solution z^* of $g(z)=0$, such that $1 \leqq |z^*| \leqq 1+\Delta$, we have $g'(z^*) \neq 0$ and

$$z_1, z_2 \in C(\beta) \Rightarrow \left|\frac{g''(z_1)}{g'(z_2)}\right| < \min\left[\left(\frac{\mu}{\beta}-1\right)\frac{2}{\beta}; \frac{\sqrt{3}-1}{\beta}\right]. \tag{15}$$

Then, Algorithm Ibstab2 *will find every such solution z^*.*

Proof. Let θ^* denote $\arg(z^*)$. Algorithm Ibstab2 generates at least one search point z_0 such that $\theta_0 \equiv \arg(z_0)$ fulfills $|\theta_0 - \theta^*| \leqq \delta\theta/2$. Now, define β as the upper limit of $|z_0 - z^*|$, when $1 \leqq |z^*| \leqq 1+\Delta$. (By Assumption 3, we need not worry about any other solutions.) It is easy to see that the assumption on η implies that

$$\beta = |(1+\Delta)\, e^{i\theta^*} - (1+\eta)\, e^{i(\theta^* \pm \delta\theta/2)}|.$$

This is β as stated in the theorem. According to Lemma 4, Algorithm Ibstab2 will now generate a call to *solve*, and, by Lemma 3 and Assumption 4, convergence follows, which proves the theorem. □

Evidently, condition (15) is impossible to check a priori. However, we will give two examples with realistic difference approximations for which $g'(z)$ and $g''(z)$ can be derived explicitly and for which condition (15) is fulfilled. This suggests that condition (15) is realistic and, thus, that Theorem 1 is acceptable as a justification of Algorithm Ibstab2.

Note that we can choose μ such that the crucial limit in (15) will be $(\sqrt{3}-1)/\beta$. In the current implementation of Ibstab2, we have $\beta = 0.11$ and $(\sqrt{3}-1)/\beta \approx 6.54$. With these numbers in mind, we turn to the examples.

Example 1. We study the one-dimensional, scalar model problem

$$u_t = u_x, \qquad 0 \leqq x < \infty, \quad 0 \leqq t.$$

At interior points, we apply the Crank-Nicolson approximation. At the boundary $x=0$, we take the one-sided approximation $u_0^{n+1} - u_0^n = \lambda(u_1^n - u_0^n)$. This yields

$$P(z, \kappa, \lambda) = \frac{\lambda}{4}(z+1)(\kappa^2 - 1) + \kappa(1-z),$$

$$g(z, \kappa, \lambda) = z - 1 + \lambda(1-\kappa).$$

(The set $M_<(z, \lambda)$ will only contain one element and thus we can omit the subscript on κ in (9).) From (6) we get

$$\kappa(z) = \frac{2}{\lambda}\frac{z-1}{z+1}[1 \pm \varphi(z)]$$

where

$$\varphi(z) = \sqrt{1 + \frac{\lambda^2}{4}\left(\frac{z+1}{z-1}\right)^2}.$$

The approximation is unstable for $\lambda > 2$. The critical solution is $z^* = 3 - 2\lambda$. Thus, we consider the case $\lambda \approx 2$, $z \approx -1$, which implies

$$\varphi(z) = 1 + \frac{\lambda^2}{8}\left(\frac{z+1}{z-1}\right)^2 + o((z+1)^4)$$

and the element of $M_<(z, \lambda)$ is

$$\kappa(z) = -\frac{\lambda}{4}\frac{z+1}{z-1} + o((z+1)^3).$$

We obtain

$$g(z) = z - 1 + \lambda + \frac{\lambda^2}{4}\frac{z+1}{z-1} + o((z+1)^3),$$

$$g'(z) = 1 - \frac{\lambda^2}{2(z-1)^2} + o((z+1)^2),$$

$$g''(z) = \frac{\lambda^2}{(z-1)^3} + o((z+1)).$$

We wish to study $|g''(z_1)|/|g'(z_2)|$ for $z_j = z^* + \delta_j e^{i\theta_j}$, where $0 \leqq \delta_j \leqq \beta \approx 0.11$, $j = 1, 2$. Using the expressions above we get

$$g''(z_1) = \frac{\lambda^2}{8(1-\lambda)^3} + o(\delta_1),$$

$$g'(z_2) = 1 - \frac{\lambda^2}{8(1-\lambda)^2} + o(\delta_2),$$

and finally

$$\frac{|g''(z_1)|}{|g'(z_2)|} = |\psi(\lambda)| + o(\delta_1) + o(\delta_2),$$

where

$$\psi(\lambda) = \frac{\lambda^2}{8(1-\lambda)^3 - \lambda^2(1-\lambda)}.$$

For $\lambda \approx 2$, we have that $\psi(\lambda) \approx -1$ and consequently, the condition of Theorem 1 is fulfilled when Algorithm Ibstab2 approaches the critical solution $\lambda \approx 2$, $z^* = 3 - 2\lambda$.

Example 2. Consider the same pde problem as in Example 1, but with the leapfrog approximation at interior points. At the boundary $x = 0$, we extrapolate, $u_0^{n+1} = u_1^{n+1}$. This yields

$$P(z, \kappa, \lambda) = (z^2 - 1)\kappa - \lambda z(\kappa^2 - 1), \qquad g(z, \kappa, \lambda) = \kappa - 1.$$

From (6), we get that the only element of $M_<(z, \lambda)$ is

$$\kappa(z) = \frac{z^2-1}{2\lambda z} + \sqrt{\left[\frac{z^2-1}{2\lambda z}\right]^2 + 1}.$$

The approximation is unstable for all λ, the critical solution being $z^* = -1$. Thus, we consider the case $z \approx -1$, which implies

$$\kappa(z) = \frac{z^2-1}{2\lambda z} + 1 + \frac{1}{2}\left[\frac{z^2-1}{2\lambda z}\right]^2 + o((z^2-1)^4).$$

We obtain

$$g(z)=\frac{z^2-1}{2\lambda z}+1+\frac{1}{2}\left[\frac{z^2-1}{2\lambda z}\right]^2+o((z^2-1)^4),$$

$$g'(z)=\frac{z^2+1}{2\lambda z^2}+\frac{z^4-1}{4\lambda^2 z^3}+o((z^2-1)^3),$$

$$g''(z)=\frac{-1}{\lambda z^3}+\frac{z^4+3}{4\lambda^2 z^4}+o((z^2-1)^2).$$

Proceeding as in Example 1, with $z_j=z^*+\delta_j e^{i\theta_j}$, $j=1,2$, we get

$$g''(z_1)=\frac{\lambda+1}{\lambda^2}+o(\delta_1),\qquad g'(z_2)=\frac{1}{\lambda}+o(\delta_2),$$

and finally

$$\frac{|g''(z_1)|}{|g'(z_2)|}=|\psi(\lambda)|+o(\delta_1)+o(\delta_2),$$

where

$$\psi(\lambda)=1+\frac{1}{\lambda}.$$

The smallest value of λ used in the implementation of Algorithm Ibstab2 is $\lambda_1=0.1$. In practice, Ibstab2 does find the instability for λ_1. The sufficient condition of Theorem 1 is fulfilled for $\lambda>0.19$, which shows that the limit $(\sqrt{3}-1)/\beta$ is of a reasonable size. Under the assumptions of Theorem 1, Ibstab2 is guaranteed to find the instability for $\lambda\approx 0.2$, which is quite sufficient. With such a low stability limit, the conclusion would be that the approximation is useless for practical purposes.

4.2. The adjustable parameters. The adjustable parameters in Algorithm Ibstab2 are $\delta\theta$, Δ, η, μ, and λ_1. From their definitions it is clear that $\delta\theta$, Δ, and η should be small; finding suitable values by experiments is straightforward. Once the values of these three parameters have been settled, Theorem 1 gives guidelines for the choice of μ.

The last parameter λ_1 should also be small because we want $Z(\lambda_1)$ to be a small perturbation of $Z(0)$. However, $Z(0)$ has few distinct elements that are typically roots of one. Thus, for *very* small λ, the elements of $Z(\lambda)$ will form clusters around the elements of $Z(0)$. Evidently, if one of the elements of such a cluster should be critical, while the others are not, then it would be very difficult for *solve* to distinguish precisely *that* element from all the surrounding ones. (As an example, take Test problem III of [16]. A closer investigation of that problem shows that for $\lambda=0.015$ there is a cluster of 18 different solutions with $z\approx -1$. One of these shows that the difference approximation is unstable, while the others are harmless.) This inherent difficulty imposes a lower limit on the choice of λ_1.

In the current implementation of Ibstab2, the parameter values are

$$\lambda_1=\delta\theta=\Delta=0.1,\qquad \eta=0.001,\qquad \mu=0.32.$$

This set of values have been used in all tests and they make the algorithm work satisfactorily.

5. Additional comments.

5.1. Multiple eigenvalues κ. In the previous sections, some details were omitted for the sake of clarity. They will now be treated.

The first point of interest is that the exact form of the general solution (7) depends on the multiplicities of the distinct elements of $M_<(z, \xi, \lambda)$. This affects $g(z)$ and implies that not only its value but also its *structure* depends on z.

To give a detailed description of (7), we introduce some concepts from matrix theory, following Zurmühl [17]. Let M be a matrix with m linearly independent eigenvectors $\phi_\nu^{(1)}$, $\nu = 1, \cdots, m$, and corresponding eigenvalues κ_ν, $\nu = 1, \cdots, m$. The chain of principal vectors corresponding to κ_ν is defined by

$$(M - \kappa_\nu I)\phi_\nu^{(\tau)} = \phi_\nu^{(\tau-1)}, \qquad \tau = 2, \cdots, e_\nu.$$

The number e_ν is the essential multiplicity of κ_ν. Using these concepts, we now state without proof Lemma 5.

LEMMA 5. *The solution of the ordinary difference equation*

$$\sum_{\nu=-l}^{r} A_\nu v_{j+\nu} = 0,$$

where A_ν are $d \times d$-matrices, A_r is nonsingular, and $v_{j+\nu}$ are $d \times 1$-vectors, is given by the d uppermost elements of the column vector w_j, where

$$w_j = \sum_{\nu=1}^{m} \Psi_\nu(j)\kappa_\nu^j.$$

Here, the scalar quantities κ_ν are the eigenvalues, counted with geometric multiplicity, of the $d(r+l) \times d(r+l)$-matrix

$$\begin{pmatrix} -A_r^{-1}A_{r-1} & \cdots & \cdots & \cdots & -A_r^{-1}A_{-l} \\ I & 0 & \cdots & \cdots & 0 \\ 0 & \ddots & 0 & \cdots & 0 \\ \vdots & & \ddots & \ddots \quad \ddots & \vdots \\ 0 & \cdots & 0 & I & 0 \end{pmatrix}.$$

The coefficients $\Psi_\nu(j)$ are vectors given by

$$\Psi_\nu(j) = \sum_{\tau=1}^{e_\nu} \sigma_{\nu\tau}\phi_\nu^{(\tau)} j^{\tau-1},$$

where $\sigma_{\nu\tau}$ are scalar coefficients, $\phi_\nu^{(\tau)}$ are the eigenvector and principal vectors corresponding to κ_ν, and e_ν is the essential multiplicity of κ_ν.

Using Lemma 5, we can study the detailed structure of $g(z)$. The fact that this structure depends on z is a problem when the systems' view of (6), (9) is applied. When the system is solved by some iterative process, the equation (9) might change its form as the iteration proceeds. A solution to this difficulty is given by the following lemma.

LEMMA 6. *Let $g_1(z)$ denote $g(z)$ in the case when all eigenvalues $\kappa_\nu(z) \in M_<(z, \xi, \lambda)$ have essential multiplicity one. Furthermore, assume that for $z = z_0$ at least one eigenvalue $\kappa_\nu(z_0) \in M_<(z_0, \xi, \lambda)$ has essential multiplicity greater than one. Then $g_1(z_0) = 0$.*

Before we prove this, let us discuss how to use it. The lemma implies that the iterative process can be applied to (6), (9), taking $g \equiv g_1$ during the entire process. If convergence is obtained to a point $(z^*, \kappa_1^*, \cdots, \kappa_N^*)$, then, by examining the multiplicities of $\kappa_1^*, \cdots, \kappa_N^*$, we can obtain the appropriate form g_* of g, corresponding to z^*. If $g_*(z^*) = 0$, then we have found a solution to (6), (9).

In the implementation of Ibstab2, $g = g_1$ is used when *solve* is called. However, in the two-dimensional case it is too complicated to investigate the essential multiplicities in detail, so in case of convergence we only check if some of the components κ_ν^* *might* have multiplicity greater than one and if so, issue a warning. In the one-dimensional case (i.e., $\xi = 0$), multiplicity two is properly treated, which is sufficient for centered, fourth-order difference approximations in space. Finally, we wish to remark that, to our knowledge, there is no report in the literature of a case where a solution to (6), (9) contains a multiple eigenvalue κ_ν^*. Hopefully, such cases are rare exceptions.

We now prove Lemma 6.

Proof. Recall that $g \equiv \det(G)$. From (8), we have that the elements of G are

$$G_{\mu\nu} = \sum_{\tau=-1}^{s} z^{n-\tau} \sum_{k=-k_1}^{k_2} e^{i\xi k} \sum_{j=0}^{q} B_{\mu jk}^{(\tau)} \Phi_\nu(j) \kappa_\nu^j,$$

$$\mu = 0, \cdots, l-1, \quad \nu = 1, \cdots, N.$$

Here, Φ_ν are $d \times 1$-vectors and $N = dl$. From Lemma 5, we have that if κ_ν is a simple eigenvalue, then $\Phi_\nu(j)$ contains the d uppermost components of the corresponding eigenvector. Let G_1 be the matrix obtained when all κ_ν are simple. If the point $(z_0, \kappa_1(z_0), \cdots, \kappa_N(z_0))$ is inserted into G_1, then two of the κ-values will coincide and correspond to the same eigenvector, which implies that two columns of G_1 are equal. Thus, $G_1(z_0, \kappa_1(z_0), \cdots, \kappa_N(z_0), \xi, \lambda)$ is singular, which proves the lemma. $\square$

5.2. Remarks on the implementation. There are also some additional implementation details that should be mentioned.

First, the perturbation analysis that is necessary if (6), (9) has a solution with $|z| = 1$ is included in the implementation. Furthermore, in the one-dimensional case the implementation distinguishes between weak and strong stability according to Gustafsson, Kreiss, and Sundström [7]. The implementation also contains a simple algorithm for checking the von Neumann stability condition.

Second, in case of an unstable difference approximation Algorithm Ibstab2 produces two values: λ_R, the value of λ for which the instability was discovered; and λ_L, the largest value of λ for which no instability was found. The implementation then proceeds with a refinement phase, based on interval bisection, until a pair λ_L, λ_R is obtained for which $\lambda_R - \lambda_L \leqq \varepsilon$, where ε is a predefined tolerance (currently $\varepsilon = 0.025$). Due to the discussion in § 4.3, no refinement is made if $\lambda_R = \lambda_1$.

Finally, in $solve(z_0, \kappa_1, \cdots, \kappa_N)$, the determinant g is scaled by a local scaling factor:

$$scale \leftarrow 1/|g(z_0)|.$$

This has been found necessary to make the solution process robust.

6. Experimental verification. Algorithm Ibstab2 has been successfully tested on a number of difference approximations. There are two possible ways to perform the tests. The first is to apply Ibstab2 to problems with known stability limits. The second is to check the results afterwards by performing numerical experiments with the difference approximation, using $\lambda \approx \lambda_L$ and $\lambda \approx \lambda_R$, respectively ($\lambda_L$ and λ_R were defined in the previous section).

Otto and Thuné reported on a large number of tests based on the second strategy [13]. The interior approximation was the same in all those cases: a centered, second-order approximation in space and a Runge-Kutta type scheme in time. The resulting difference method needed three boundary conditions at each boundary. Several sets of numerical boundary conditions were analyzed. Furthermore, we considered grid overlaps: Ibstab2 was used to investigate the stability of a number of overlapping conditions. The results produced by Ibstab2 were all experimentally demonstrated to be correct. The stability limits that were obtained are of independent interest and the article shows the intended way of using Ibstab2 in realistic applications.

Here, we will discuss some test problems with known results. Below, the problems are presented in a uniform way. Some explanations are needed. The computational domain is $0 \leqq t$, $0 \leqq x < \infty$ and, in the two-dimensional case, $-\infty < y < \infty$. The partial difference equations ($p\Delta e$) are presented using the usual difference operators. The argument to these operators is the stepsize, but whenever a full step (Δx or Δy) is assumed, the argument is omitted. In some test problems, two alternative boundary conditions, marked by (a) and (b), are included, giving two different test cases. Finally, we have pointed out the cases where the boundary conditions do not decrease the stability of the approximation. In these cases, the explicit stability condition contains the necessary von Neumann limit, which we denote by λ_{von}.

TEST PROBLEM 1.

Pde: $\begin{pmatrix} u \\ v \end{pmatrix}_t = \begin{pmatrix} 0 & a(x) \\ a(x) & 0 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}_x, \qquad a(0) = 1$

PΔe: Leap-Frog

Boundary condition(s): $u_0^{n+1} = 0$, $v_0^{n+1} = 2v_1^{n+1} - v_2^{n+1}$

Explicit stability condition: $0 < \lambda < \lambda_{\text{von}} = 1$, weak stability

Reference(s): [6].

TEST PROBLEM 2.

Pde: $u_t = a(x, t)u_x$, (a) $a(0, t) = -1$, (b) $a(0, t) = 1$

PΔe: Leap-Frog (4, 2):

$$u_j^{n+1} = u_j^{n-1} + 2\Delta t a_j^n \left(I - \frac{h^2}{6} D_{+,x} D_{-,x} \right) D_{0,x} u_j^n$$

Boundary condition(s):

(a) $u_0^{n+1} = 0$,

$$u_1^{n+1} = u_1^{n-1} + \frac{\lambda}{3} \left(-2u_0^n - \frac{3}{2}(u_1^{n+1} + u_1^{n-1}) + 6u_2^n - u_3^n \right)$$

(b) $$u_0^{n+1} = u_0^{n-1} + \frac{\lambda}{3} \left(-\frac{11}{2}(u_0^{n+1} + u_0^{n-1}) + 18u_1^n - 9u_2^n + 2u_3^n \right)$$

$$u_1^{n+1} = u_1^{n-1} + \frac{\lambda}{3} \left(-2u_0^n - \frac{3}{2}(u_1^{n+1} + u_1^{n-1}) + 6u_2^n - u_3^n \right)$$

Explicit stability condition: (a) $0 < \lambda < \lambda_{\text{von}} \approx 0.7287$, (b) The numerical investigation in [14] carried out for discrete values of λ, showed instability for $\lambda \geqq 0.68$

Reference(s): [11], [14].

TEST PROBLEM 3.

Pde: $u_t = a(x, t)u_x, \qquad a(0, t) = 1$

PΔe: Mesh refinement near the boundary $x = 0$, with mesh refinement factor S. We use $S = 5$. On the refined mesh, Leap-Frog is used. On the coarse grid, we take Leap-Frog (4, 2) (cf. Test Problem 2).

Boundary condition(s): $u_0^{n+1}=u_0^{n-1}+2\lambda_f(u_1^n-\frac{1}{2}(u_0^{n+1}+u_0^{n-1}))$, where λ_f is the value of λ on the fine grid.

Explicit stability condition: Unconditionally unstable for $S\geqq 2$

Reference(s): [12].

TEST PROBLEM 4.

Pde: The linearized Euler equations:

$$\begin{pmatrix}\rho\\ u\\ p\end{pmatrix}_t+\begin{pmatrix}\hat{u} & \hat{\rho} & 0\\ 0 & \hat{u} & \hat{\rho}^{-1}\\ 0 & \gamma\hat{p} & \hat{u}\end{pmatrix}\begin{pmatrix}\rho\\ u\\ p\end{pmatrix}_x=0.$$

We assume subsonic inflow, i.e., $0<\hat{u}<c$, where c is the local speed of sound.

PΔe: Crank-Nicolson

Boundary condition(s): $\rho_0^{n+1}=0,\qquad u_0^{n+1}=0,$
$[\hat{\rho}cu-p]_0^{n+1}=2[\hat{\rho}cu-p]_1^n-[\hat{\rho}cu-p]_2^{n-1}$

Explicit stability condition: In the test, $\hat{u}$, $\hat{\rho}$, and c were chosen such that the stability condition was $0<\lambda\leqq 2$.

Reference(s): [8].

TEST PROBLEM 5.

Pde: $u_t=u_x+u_y$

PΔe: Euler backwards, split form: $(I-\Delta t\,D_{0,x})(I-\Delta t\,D_{0,y})u_{j,l}^{n+1}=u_{j,l}^n$

Boundary condition(s): (a) $u_{0,l}^{n+1}=2u_{1,l}^{n+1}-u_{2,l}^{n+1}$, (b) $u_{0,l}^{n+1}=2u_{1,l+1}^{n+1}-u_{2,l+2}^{n+1}$

Explicit stability condition: (a) $0<\lambda<\lambda_{\text{von}}=\infty$, (b) Unconditionally unstable

Reference(s): [2].

TEST PROBLEM 6.

Pde: $u_t=u_x+u_y$

PΔe: Burstein:

$$\begin{aligned}u_{j,l}^{n+1/2}&=\frac{1}{4}(u_{j+1/2,l}^n+u_{j-1/2,l}^n+u_{j,l+1/2}^n+u_{j,l-1/2}^n)\\&\quad+\frac{\Delta t}{4}\left[D_{0,x}\left(\frac{\Delta x}{2}\right)(u_{j,l+1/2}^n+u_{j,l-1/2}^n)\right.\\&\quad\left.+D_{0,y}\left(\frac{\Delta y}{2}\right)(u_{j+1/2,l}^n+u_{j-1/2,l}^n)\right]\\u_{j,l}^{n+1}&=u_{j,l}^n+\frac{\Delta t}{2}\left[D_{0,x}\left(\frac{\Delta x}{2}\right)(u_{j,l+1/2}^{n+1/2}+u_{j,l-1/2}^{n+1/2})\right.\\&\quad\left.+D_{0,y}\left(\frac{\Delta y}{2}\right)(u_{j+1/2,l}^{n+1/2}+u_{j-1/2,l}^{n+1/2})\right]\end{aligned}$$

Boundary condition(s): (a) $u_{0,l}^{n+1}=2u_{1,l}^{n+1}-u_{2,l}^{n+1}$, (b) $u_{0,l}^{n+1}=2u_{1,l+1}^{n+1}-u_{2,l+2}^{n+1}$

Explicit stability condition: (a) $0<\lambda<\lambda_{\text{von}}=1/\sqrt{2}$,
(b) Unconditionally unstable

Reference(s): [2].

TEST PROBLEM 7.

Pde: $\begin{pmatrix}u\\ v\end{pmatrix}_t=\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}\begin{pmatrix}u\\ v\end{pmatrix}_x+\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}\begin{pmatrix}u\\ v\end{pmatrix}_y$

PΔe: Leap-Frog

Boundary condition(s): $u_{0,l}^{n+1} = 2u_{1,l}^{n} - u_{2,l}^{n-1} \qquad v_{0,l}^{n+1} = 0$

Explicit stability condition: In the reference, a numerical investigation was made for discrete values $\lambda \in]0, 0.5]$. For those values no instability was found.

Reference(s): [1].

TEST PROBLEM 8.

Pde: $u_t = u_x + u_y$

PΔe: Leap-Frog

Boundary condition(s): $u_{0,l}^{n+1} = u_{0,l}^{n} + \Delta t(D_{+,x} + D_{0,y})u_{0,l}^{n}$

Explicit stability condition: $0 < \lambda \leqq 0.4$

Reference(s): [1].

In all the test cases, Ibstab2 gave correct stability limits. It is also of interest to analyze the performance of the algorithm with respect to the following parameters:

n_λ—the number of λ-values that were treated,
$\bar{n}_z$—the average number of z-values per λ-value,
$\bar{n}_\xi$—the average number of ξ-values per λ-value,
$\bar{n}_s$—the average number of *solve*-calls per λ-value,
t—the total cpu-time (in seconds),[4]
t_s—the cpu-time spent in *solve* (in seconds),
$\bar{t}$—the average cpu-time per z-value, disregarding the time spent in solve, $\bar{t} = (t - t_s)/(n_\lambda \bar{n}_z)$,
$\bar{t}_s$—the average cpu-time per *solve*-call.

The test results in terms of these parameters are collected in Table 1.

Test Problems 1-4 are the one-dimensional problems presented in [16]. They are included here to make it possible to compare the new version with the earlier one. The results for the early version are given in parentheses.[5]

The most striking difference is the substantial decrease in the number of *solve*-calls. The results in Table 1 show that in the early version of Ibstab, *solve* was called at

TABLE 1
Results of Ibstab2 *tests. The notation is explained in the text.*

Problem	n_λ	$\bar{n}_\xi$	$\bar{n}_z$	$\bar{n}_s$	t	t_s	$\bar{t}_s/\bar{t}$
1	9 (22)	-	42 (67)	7 (28)	35	33	99.0
2(a)	8 (16)	-	39 (71)	3 (31)	15	3	3.2
2(b)	7 (15)	-	49 (128)	21 (118)	29	11	1.4
3	1 (3)	-	31 (57)	6 (54)	3	1	2.6
4	14 (33)	-	56 (117)	11 (99)	81	74	53.8
5(a)	6	32	2016	54	76	13	7.7
5(b)	1	17	1009	66	16	10	25.5
6(a)	8	16	1008	31	55	14	11.1
6(b)	1	15	883	48	15	8	21.0
7	8	16	1008	208	1511	1209	19.4
8	8	110	6948	637	610	303	10.8

[4] As we wish to study the performance of Algorithm Ibstab2, the quantity t does *not* include the time spent in the procedure for checking the von Neumann condition (cf. § 5).

[5] However, the timings are not comparable. The quantity t_s was not measured in the early tests. As for t, the early tests were run on a BASF 7/68 computer, using the WATFIV compiler, whereas the new tests were executed on a MicroVax II using the standard Vax FORTRAN compiler. In both cases the nonoptimizing compiler mode was used.

42–95 percent of the search points. The corresponding figures for Ibstab2, including the two-dimensional cases, are 2–43 percent. The average is 12 percent for Ibstab2. This is important, as is shown by a comparison between $\bar{t}$ and $\bar{t}_s$ for all the test problems. From the last column of Table 1, we see that in general one call to *solve* is 3–30 times as expensive as the treatment of one z-value when no *solve*-call is made. Thus, keeping the number of calls to *solve* low is an important way of making the algorithm efficient.

For the two-dimensional problems, Test Problems 5–8, we see that the time for a complete analysis ranges from 16 seconds, for the unconditionally unstable case 5(b), to 25 minutes for Test Problem 7. Much of the time difference between Test Problem 7 and, e.g., Test Problem 6(a), for which we have the same number of search points, is explained by the difference in the number of calls to *solve*. However, there is also a difference in $\bar{t}$, because Test Problem 7 is a two-dimensional pde *system*. For a system, the polynomials in (6) and (9) are more expensive to evaluate than for a scalar problem. This affects the quantity $\bar{t}$, which essentially measures the time for operations consisting of evaluations of these polynomials and their derivatives. For Test Problem 7, we have that $\bar{t} \approx 0.04$ s, whereas for the scalar two-dimensional test problems $\bar{t} \approx 0.006$ s.

In general, the cpu time will increase with the number of equations involved in the pde problem and with the order of accuracy of the difference approximation. For problems with many equations and/or high order of accuracy we might have to consider a parallelized version of Ibstab2.

7. Conclusions. The ultimate goal of the Ibstab project is a complete problem solving environment, combining symbolic and numerical routines, for the stability analysis of difference methods for hyperbolic systems. A first sketch of such an environment was presented in [16]. The conclusion of the present article is that we can now consider the numerical part of the project to be essentially finished. The new Algorithm Ibstab2 can handle two-dimensional problems and could easily be generalized to higher-dimensional problems. Furthermore, it is much more efficient than the earlier one-dimensional algorithm, due to improvements of implementational details. To increase the speed further we would have to consider a parallelized version of the algorithm.

Continued work in the Ibstab project will concern the symbol manipulation routines. In the pilot version these were written in Lisp and had a black box design. The user gave a description of the difference approximation. The system (6), (9) was then automatically generated as Fortran routines. A future version should be written in some symbolic algebraic manipulations language, e.g., Reduce, and the black box design ought to be abandoned. A flexible tool box design would be preferable. For example, the environment should include tools for checking the convenient stability conditions of Goldberg and Tadmor.

REFERENCES

[1] S. Abarbanel and D. Gottlieb, *Stability of two dimensional initial boundary value problems using Leap Frog type schemes*, ICASE Report No. 78-6, NASA Langley Research Center, Hampton, VA, 1978.

[2] S. S. Abarbanel and E. M. Murman, *Stability of two-dimensional hyperbolic initial boundary value problems for explicit and implicit schemes*, J. Comput. Phys., 48 (1982), pp. 160–167.

[3] K. M. Brown, *Computer oriented algorithms for solving systems of simultaneous nonlinear algebraic equations*, in Numerical Solution of Systems of Nonlinear Algebraic Equations, G. A. Byrne and C. A. Hall, eds., Academic Press, New York, 1973.

[4] W. M. COUGHRAN, JR., *On the approximate solution of hyperbolic initial-boundary value problems*, Ph.D. thesis, Report No. STAN-CS-80-806, Department of Computer Science, Stanford University, Stanford, CA, 1980.

[5] M. GOLDBERG AND E. TADMOR, *Convenient stability criteria for difference approximations of hyperbolic initial-boundary value problems*. II, Math. Comp., 48 (1987), pp. 503-520.

[6] B. GUSTAFSSON, *The convergence rate for difference approximations to mixed initial-boundary value problems*, Math. Comp., 29 (1975), pp. 396-405.

[7] 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.

[8] B. GUSTAFSSON AND J. OLIGER, *Stable boundary approximations for implicit time discretizations for gas dynamics*, SIAM J. Sci. Statist. Comput., 3 (1982), pp. 408-421.

[9] D. MICHELSON, *Stability theory of difference approximations for multidimensional initial-boundary value problems*, Math. Comp., 40 (1983), pp. 1-45.

[10] *Nag Library Manual, Mark* 12, Numerical Algorithms Group Limited, Oxford, 1987.

[11] J. OLIGER, *Fourth order difference methods for the initial boundary-value problem for hyperbolic equations*, Math. Comp., 28 (1974), pp. 15-25.

[12] ———, *Hybrid difference methods for the initial boundary-value problems for hyperbolic equations*, Math. Comp., 30 (1976), pp. 724-738.

[13] K. OTTO AND M. THUNÉ, *Stability of a Runge-Kutta method for the Euler equations on a substructured domain*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 154-174.

[14] D. M. SLOAN, *Boundary conditions for a fourth order hyperbolic difference scheme*, Math. Comp., 41 (1983), pp. 1-11.

[15] M. THUNÉ, IBSTAB—*A software system for automatic stability analysis of difference methods for hyperbolic initial-boundary value problems*, Ph.D. thesis, Report No. 93, Department of Computer Sciences, Uppsala University, Uppsala, Sweden, 1984.

[16] ———, *Automatic* GKS *stability analysis*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 959-977.

[17] R. ZURMÜHL, *Matrizen*, Springer-Verlag, Berlin, New York, 1964.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 1, pp. 82-91, January 1990

"MISSING" BOUNDARY CONDITIONS? DISCRETIZE FIRST, SUBSTITUTE NEXT, AND COMBINE LATER*

ARTHUR E. P. VELDMAN†

Abstract. A simple approach exists to prevent the need for constructing boundary conditions in situations where they are not explicitly supplied by the original analytical formulation of the problem. An example is the Poisson equation for the pressure in calculations of incompressible flow. Other examples are the streamfunction-vorticity formulation where no condition for the vorticity is present, and ADI methods where boundary conditions for the intermediate timesteps must be provided. In short, this approach can be described as follows: first discretize the equations of motion, next substitute the original boundary conditions (for the velocity), and finally combine the discrete equations (e.g., to a modified Poisson equation).

Key words. boundary conditions, discretization methods, incompressible Navier-Stokes equations

AMS(MOS) subject classifications. 65N05, 76D05

1. Introduction.

When the incompressible Navier-Stokes equations are solved numerically, often boundary conditions seem to be "missing." One example is formed by the boundary conditions for the pressure when a Poisson equation is employed. Another example is the boundary condition for the vorticity in case a streamfunction-vorticity formulation is used. Uncertainty exists about the choice of these conditions; when Neumann conditions are selected, the corresponding compatibility relation poses an additional difficulty.

Gresho and Sani [1] give an extensive discussion of the former example. They discuss a number of approaches used to solve the above problem. Their favorite approach is what they call the "direct attack." This is a simple method that has been known for at least two decades (see the references in Chapter 6.3.1 of [2]). Gresho and Sani show that this approach circumvents the problem of the "missing" boundary conditions in a natural way.

From discussions with colleagues it became clear that this "direct attack" is applicable to many more situations where boundary conditions are "missing." Therefore in this paper we want to highlight this approach and show some applications. It will not be surprising that the methods being obtained in this way are familiar ones. However, the way in which they have been derived ensures there is no need to distrust them, whereas other derivations of the same formulas might leave some room for distrust.

The starting point is an analytical set of equations, including boundary conditions, that is well posed and for which a unique solution exists. For the unsteady incompressible Navier-Stokes equations we may use its formulation in primitive variables (velocity and pressure). At solid walls only boundary conditions for the velocity are required to make the solution unique [3, Chap. 3, § 3]. No conditions on the pressure have to be prescribed in the continuum case. The Navier-Stokes equations will be discretized and its boundary conditions substituted. Hereafter the discrete set of equations may be combined in any way that is found convenient, e.g., to a discrete Poisson equation or to a discrete streamfunction-vorticity formulation. This shuffling of the equations does not change the solution, and hence is harmless. In short, this approach can be described as: discretize first, substitute next, and combine later.

* Received by the editors June 17, 1988; accepted for publication (in revised form) November 28, 1988.

† Faculty of Technical Mathematics and Informatics, Delft University of Technology, P.O. Box 356, 2600 AJ Delft, the Netherlands.

For those who are unfamiliar with this approach we will present it in detail for the pressure conditions. Moreover, a number of other applications will be given. Next to the streamfunction-vorticity formulation for the Navier–Stokes equations, we apply it to the shallow-water equations and to ADI methods. In Appendix A it also will be shown useful for the treatment of the constraints in differential-algebraic equations.

2. Problem. Consider a rectangular domain Ω with boundary Γ on which the incompressible Navier–Stokes equations have to be solved:

$$\operatorname{div} \mathbf{q} = 0, \tag{2.1a}$$

$$\frac{\partial \mathbf{q}}{\partial t} + (\mathbf{q} \cdot \operatorname{grad})\mathbf{q} = -\operatorname{grad} p + \nu \operatorname{div} \operatorname{grad} \mathbf{q}, \tag{2.1b}$$

with boundary conditions

$$\mathbf{q} = \mathbf{q}^{\Gamma} \quad \text{on } \Gamma \tag{2.2}$$

(often $\mathbf{q}^{\Gamma} = 0$). Here $\mathbf{q} = (u, v)$ is the velocity vector, p is the kinematic pressure, and ν is the kinematic viscosity.

As the treatment of the convective terms and diffusive terms is irrelevant for the discussion that follows, the equations of motion (2.1) will be abbreviated as

$$\operatorname{div} \mathbf{q} = 0, \tag{2.3a}$$

$$\frac{\partial \mathbf{q}}{\partial t} + \operatorname{grad} p = \mathbf{R}. \tag{2.3b}$$

The above equations can be combined to obtain a Poisson equation for the pressure p:

$$\operatorname{div} \operatorname{grad} p = \operatorname{div} \mathbf{R} - \frac{\partial}{\partial t} \operatorname{div} \mathbf{q}. \tag{2.4}$$

From the analytical point of view, the second term in the right-hand side of (2.4) vanishes, but it has been retained to stress that its discrete numerical treatment is nontrivial: accumulation of errors is possible. This will be clarified in Appendix A.

Usually after this stage the equations of motion, (2.3b) and (2.4), are discretized. The latter, elliptic, equation obviously requires boundary conditions for p. These are not immediately available, since in (2.2) only the velocity appears. It is possible to derive boundary conditions for the pressure from the momentum equation (2.1b)—usually the normal component is used—but their evaluation requires values for the velocity components in points located one full mesh outside the boundary Γ. These values are not available. Various methods have been proposed as a remedy. This has led to confusion, and some controversy has arisen, as referred to above. A more complete discussion of this point is given by Gresho and Sani [1] and Peyret and Taylor [2, Chap. 6].

3. Solution. The above problem can be circumvented by first discretizing the original equations (2.1). In these discrete equations the boundary condition (2.2) is substituted. Only hereafter we will perform in a discrete sense the above reformulation. This leads to a discrete version of (2.4), but with a modification near the boundary, such that no boundary conditions are required. This process will be worked out in more detail for a discretization using the well-known staggered grid from the MAC-method [4]. The time-integration will be performed with an explicit two-level scheme, but the discussion below applies to any time-integration method. The discrete time-evolution can be written as

$$\operatorname{div}\mathbf{q}^{n+1}=0, \tag{3.1a}$$

$$\frac{\mathbf{q}^{n+1}-\mathbf{q}^{n}}{\delta t}+\operatorname{grad}p^{n+1}=\mathbf{R}^{n} \tag{3.1b}$$

where n indicates the time level. The term $\operatorname{grad}p$ is written with an index $n+1$ to stress that its value has to be such that $\operatorname{div}q^{n+1}=0$; [2, Chap. 6] uses the same convention. Equation (3.1b) can be reformulated as

$$\mathbf{q}^{n+1}=\mathbf{q}^{n}+\delta t\mathbf{R}^{n}-\delta t\operatorname{grad}p^{n+1}. \tag{3.2}$$

At this moment we do *not* substitute (3.2) into (3.1a) to create the Poisson equation. Instead, we discretize first. Let the equations in discrete form be given by

$$D_h\mathbf{q}_h^{n+1}=0, \tag{3.3a}$$

$$\frac{\mathbf{q}_h^{n+1}-\mathbf{q}_h^{n}}{\delta t}+G_hp_h^{n+1}=\mathbf{R}_h^{n} \tag{3.3b}$$

where D_h and G_h are the discrete div and grad operator, respectively. Further $\mathbf{q}_h$, p_h, and $\mathbf{R}_h$ are the discrete grid functions corresponding with $\mathbf{q}$, p, and $\mathbf{R}$. Equations (3.3) are essentially the equations that are being solved. The way in which they are solved only uses some "shuffling" of these equations.

The treatment of the continuity equation in a cell adjacent to the boundary is the only thing that matters. Consider the cell given in Fig. 1. The discrete continuity equation (3.3a) reads

$$\frac{1}{\delta x}(u_e^{n+1}-u_w^{n+1})+\frac{1}{\delta y}(v_n^{n+1}-v_s^{n+1})=0. \tag{3.4}$$

Next the boundary condition (2.2) is applied, which states that $u_e^{n+1}=u_e^{\Gamma}$. Only thereafter is the discrete version of (3.2) substituted. We end up with an equation for

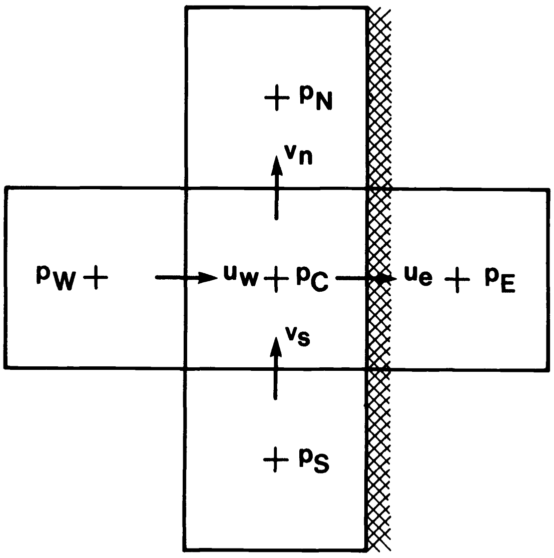


FIG. 1. *Stencil for (modified) Poisson equation near a boundary.*

the pressure, in which the pressure defined in the grid cell E, lying outside the domain, does not occur. For completeness, the pressure equation reads

$$(3.5)\quad \begin{aligned} &-\left\{\frac{1}{(\delta x)^2}+\frac{2}{(\delta y)^2}\right\}p_C^{n+1}+\frac{1}{(\delta x)^2}p_W^{n+1}+\frac{1}{(\delta y)^2}(p_N^{n+1}+p_S^{n+1}) \\ &\quad =\frac{1}{\delta t}\left\{\frac{1}{\delta x}(u_e^\Gamma-u_w^n)+\frac{1}{\delta y}(v_n^n-v_s^n)\right\}+\left\{-\frac{1}{\delta x}R_w^n+\frac{1}{\delta y}(R_n^n-R_s^n)\right\}. \end{aligned}$$

In a more compact notation, the above can be formulated as follows. Split the discrete divergence operator as defined in (3.3a) in two parts

$$(3.6a)\qquad D_h=D_h^0+D_h^\Gamma$$

where D_h^0 corresponds with velocity components defined in interior points, and D_h^Γ corresponds with velocity components defined on the boundary Γ. Then (3.3a) can be written as

$$(3.6b)\qquad D_h^0\mathbf{q}_h^{n+1}=-D_h^\Gamma\mathbf{q}_h^{n+1}.$$

The right-hand side is known from the boundary condition (2.2). Now substitute the discrete version of (3.2), i.e., (3.3b):

$$\mathbf{q}_h^{n+1}=\mathbf{q}_h^n+\delta t\mathbf{R}_h^n-\delta tG_hp^{n+1},$$

into (3.5). The result is

$$(3.7)\qquad D_h^0G_hp^{n+1}=D_h^0\left(\frac{1}{\delta t}\mathbf{q}_h^n+\mathbf{R}_h^n\right)+\frac{1}{\delta t}D_h^\Gamma\mathbf{q}_h^{n+1}.$$

We have a system of $N_x\times N_y$ equations (where N_x and N_y are the number of cells in x- and y-direction, respectively) for an equal number of unknowns p. It can be solved straightforwardly.

Remark 1. The system (3.7) is singular, since a constant pressure satisfies $G_hp=0$. However, due to the staggered grid, $p=Ct$ is the only solution of the homogeneous system. The right-hand side of (3.7) has to satisfy a compatibility relation: it must be perpendicular to the nullspace of $(D_h^0G_h)^T$. Here, this amounts to

$$(3.8)\qquad \sum_{i=1}^{N_x}\sum_{j=1}^{N_y}r_{ij}=0,$$

where r_{ij} is an abbreviation for the right-hand side of (3.7) in the cell (i,j). It is easily verified that

$$\sum_{i=1}^{N_x}\sum_{j=1}^{N_y}(D_h^0\phi)_{ij}=0$$

for any ϕ. Hence (3.8) reduces to

$$(3.9)\qquad \frac{1}{\delta y}\sum_{i=1}^{N_x}(v_{i,N_y+1/2}-v_{i,1/2})+\frac{1}{\delta x}\sum_{j=1}^{N_y}(u_{N_x+1/2,j}-u_{1/2,j})=0.$$

After multiplication with $\delta x\,\delta y$, (3.9) equals a discrete version of

$$\int_\Gamma \mathbf{q}\cdot\mathbf{n}\,ds=0,$$

which is a relation that must hold analytically. Hence any reasonable choice of the discrete boundary condition (2.2) will satisfy the compatibility condition.

Remark 2. Frequently, the full Poisson equation

$$D_h G_h p^{n+1} = D_h \left(\frac{1}{\delta t} \mathbf{q}_h^n + \mathbf{R}_h^n \right) \tag{3.10}$$

is solved with the Neumann boundary condition

$$\mathbf{n} \cdot G_h p^{n+1} = \mathbf{n} \cdot \left[\mathbf{R}_h^n - \frac{1}{\delta t} (\mathbf{q}_h^{n+1} - \mathbf{q}_h^n) \right] \quad \text{on } \Gamma. \tag{3.11}$$

The problem is that $\mathbf{R}_h^n$ cannot be computed since it requires a velocity component in a point one mesh outside the domain. This gave rise to approximations such as $\mathbf{R}_h^n = 0$ (hence $(\partial/\partial n)p = 0$), resulting in ambiguities and confusion. However, when the terms $\mathbf{R}_h^n$ that appear in (3.10) and in (3.11) are treated consistently, then the ambiguity cancels (see [2 Chap. 6.3]). Thus (3.5) can also be considered as obtained from (3.10) in which (3.11) is substituted.

Remark 3. The above technique works equally well for other time-integration methods. To see this consider the semidiscretized version of (2.3), i.e.,

$$D_h \mathbf{q}_h = 0, \qquad \frac{d}{dt} \mathbf{q}_h + G_h p_h = \mathbf{R}_h.$$

By proceeding as in (3.6a), the pressure follows from

$$D_h^0 G_h p_h = D_h^0 \mathbf{R}_h + D_h^\Gamma \frac{\partial}{\partial t} \mathbf{q}_h. \tag{3.12}$$

(To prevent error accumulation a term $(1/\delta t) D_h \mathbf{q}_h^n$ should be added to the right-hand side of (3.12).) No matter which time-integration method is used, the pressure can be computed from (3.12) without needing boundary conditions. When an ADI method is used, § 4 shows how to deal with the boundary velocities required by $\mathbf{R}_h$ at intermediate time levels.

Remark 4. In (3.3) the discrete divergence D_h and discrete gradient G_h are not yet specified. Higher-order discretizations are allowed, as long as the total number of equations in (3.3) plus the velocity boundary conditions equals the total number of unknowns $\mathbf{q}_h$ and p_h.

4. Other applications.

The philosophy presented above can be described as follows:

Step (1). Discretize the equations of motion in their original (velocity-pressure) formulation.

Step (2). Substitute the boundary conditions.

Step (3). Combine the discrete equations into the desired form.

The preceding section shows how this philosophy can be applied to prevent the need for a boundary condition for the pressure in incompressible flow computations. There are more situations where this philosophy can be applied. We will briefly describe a few of them:

- The ψ-ω formulation in incompressible flow;
- The shallow-water equations;
- ADI methods.

In Appendix A another, generalized, application is presented:

- Differential equations with algebraic constraints.

ψ-ω formulation.

$$\nu\,\Delta\omega = -\frac{\partial(\psi,\omega)}{\partial(x,y)}, \tag{4.1a}$$

$$\Delta\psi = -\omega, \tag{4.1b}$$

with homogeneous boundary conditions

$$\psi = \frac{\partial\psi}{\partial n} = 0 \quad \text{on } \Gamma. \tag{4.2}$$

Thus we have two boundary conditions for the streamfunction ψ, whereas one for the vorticity ω is "missing." Usually, the Dirichlet condition in (4.2) is added to (4.1b), while the Neumann condition is manipulated into a condition for ω which is added to (4.1a). Many ways exist to do this; (see [2, Chap. 6.5]).

As an alternative, the above philosophy can be applied.

Step (1) means that we should start with a velocity-pressure formulation that is discretized, e.g., the steady version of (3.3) from the previous section. Then a discrete streamfunction ψ_h and vorticity ω_h are defined by

$$\frac{1}{\delta y}[(\psi_h)_{i+1/2,j+1/2}-(\psi_h)_{i+1/2,j-1/2}] = u_{i+1/2,j}, \tag{4.3a}$$

$$\frac{1}{\delta x}[(\psi_h)_{i+1/2,j+1/2}-(\psi_h)_{i-1/2,j+1/2}] = -v_{i,j+1/2}, \tag{4.3b}$$

$$(\omega_h)_{i+1/2,j+1/2} = \frac{1}{\delta x}[v_{i+1,j+1/2}-v_{i,j+1/2}] - \frac{1}{\delta y}[u_{i+1/2,j+1}-u_{i+1/2,j}]. \tag{4.4}$$

Note that ψ_h and ω_h are located in the vertices of the grid cells: their familiar location. The discrete continuity equation (3.3a) is identically satisfied by the choice (4.3). Also we have $\psi_h = 0$ on Γ. Furthermore, it is easily verified that

$$\Delta_h\psi_h = -\omega_h \tag{4.5}$$

where Δ_h is the usual five-point formula. Thus far, there is nothing new.

Step (2) implies substituting the boundary conditions for the velocity $\mathbf{q}_h$ into (3.3b).

Step (3) means that we should take the discrete rotation (curl) of the discrete momentum equation (3.3b). Hence we perform

$$\begin{aligned}&\frac{1}{\delta x}\left[y\text{-equation at }\left(i+1, j+\frac{1}{2}\right) - y\text{-equation at }\left(i, j+\frac{1}{2}\right)\right]\\ &-\frac{1}{\delta y}\left[x\text{-equation at }\left(i+\frac{1}{2}, j+1\right) - x\text{-equation at }\left(i+\frac{1}{2}, j\right)\right].\end{aligned} \tag{4.6}$$

This need only be done in the interior vertices. Substituting (4.3) into (4.4), we obtain a discrete version of (4.1a). The resulting discrete Laplacian becomes the usual five-point formula; the form of the discrete convective terms depends on the discretization performed in (3.3). Important is that no boundary values of ω at Γ are required any more.

Remark 1. This approach does not lead to discretizations that were unknown thus far. We leave it to the reader to verify that the resulting equations can also be obtained when the vorticity boundary condition is chosen according to Thom's formula [2, eq. (6.5.10)]. The latter is usually derived from a Taylor expansion using (4.1b) at the boundary.

Remark 2. Equation (4.6) is an algebraic combination of the discrete equations (3.3). Thus, when the solution of (4.5) and (4.6) is expressed in u and v (using (4.3)) it is identical to the steady solution of (3.3).

Shallow-water equations. Another application of the above philosophy is formed by the shallow-water equations. In a linearized primitive-variable form they read

$$\frac{\partial \mathbf{q}}{\partial t}+(\mathbf{q}\cdot \operatorname{grad})\mathbf{q}+g \operatorname{grad} \zeta=0, \tag{4.7}$$

$$\frac{\partial \zeta}{\partial t}+(\mathbf{q}\cdot \operatorname{grad})\zeta+H \operatorname{div} \mathbf{q}=0. \tag{4.8}$$

Here $\mathbf{q}$ is the depth-integrated velocity vector, ζ is the surface elevation, H is the linearized water height, and g is the gravitational acceleration. Boundary conditions are often formulated in terms of the velocity components; no condition for ζ exists then.

These equations are discretized in a staggered arrangement, as in the MAC-method. The elevation ζ is defined in cell centers, as is the pressure p. There is more similarity between ζ and p: the momentum equation (4.7) contains grad ζ, while the continuity equation (4.8) contains div $\mathbf{q}$. As above, a Poisson-type equation can be derived. It is an unsteady wave equation that reads

$$\frac{\partial^2 \zeta}{\partial t^2}-gH \operatorname{div} \operatorname{grad} \zeta=\mathrm{RHS}$$

where RHS contains all convective terms.

When this equation is used in the computation, a boundary condition for ζ is required. This one is "missing" however, but the above philosophy can be used to circumvent the problems. Due to the similarity between p and ζ, this proceeds in a way similar to that described in § 3.

ADI-methods. Consider a semidiscretized equation

$$\frac{d\phi}{dt}=D\phi[=(D_x+D_y)\phi] \tag{4.9}$$

where the spatial differential operator D can be split into one in x-direction (D_x) and one in y-direction (D_y). When (4.9) is solved by an ADI-method, or more generally a splitting-up (fractional step) method [5], conditions are required for the boundary values of ϕ at intermediate time levels. Consider, for instance, the Peaceman-Rachford method (in Douglas-Gunn notation) in two dimensions [2, Chap. 2.8]:

$$(I-\tfrac{1}{2}\delta t\, D_x)(\phi^*-\phi^n)=\delta t\, D\phi^n, \qquad (I-\tfrac{1}{2}\delta t\, D_y)(\phi^{n+1}-\phi^n)=\phi^*-\phi^n. \tag{4.10}$$

The term $D_x\phi^*$ will, in general, require values of ϕ^* on the boundary that are not immediately available. When the boundary conditions for ϕ are independent of t there is no problem, as the above splitting is time-consistent. But in other cases the boundary values of ϕ^* have to follow from nontrivial computations (see, e.g., Mitchell and Griffiths [6]).

The above philosophy suggests first to substitute the boundary conditions in (4.9). Hereafter the splitting is performed. This will lead to slightly modified D_x and D_y for which no boundary values are required. Hence the problem of the missing boundary values for the intermediate time levels is solved. Again, this strategy is not new: we have recovered Marchuk's approach for treating the intermediate time levels [5].

Marchuk writes his motivation in words that exactly describe the philosophy presented in this paper. Concluding this section, we can do no better than cite him [5, p. 410]:

... it is much simpler first to put the original problem of mathematical physics into correspondence with a system of difference equations (with respect to the spatial variables) and then to eliminate the boundary conditions using the difference analogs of the boundary conditions, the accuracy of which matches that of the difference equations. Having done this, we can next proceed by approximating the equations in time using the splitting-up method or another algorithm. This approach allows us to sidestep the compatibility problem for the boundary conditions....

5. Conclusion. The paper describes a philosophy that can be used when boundary conditions are "missing." It can be formulated in short as: discretize first, substitute next, and combine later. The philosophy is not new, and neither are the resulting methods. But apparently its power is not yet generally appreciated, as the discussions that pop up now and then in the literature reveal. Four applications of the philosophy have been presented. It is hoped that these will help to enlarge the acquaintance with this solution to the problem of the "missing" boundary conditions.

Appendix A. The treatment of the constraint $\operatorname{div} \mathbf{q} = 0$ requires some care in the time-integration method. Accumulation of errors is possible. In essence this is due to the difference that exists numerically between

$$\operatorname{div} \mathbf{q} = 0 \tag{A1}$$

and

$$\frac{\partial}{\partial t} \operatorname{div} \mathbf{q} = 0 \tag{A2}$$

with a homogeneous initial condition. We will point out this difference using a formulation in which only a time-discretization is used. It equally applies to the space-discretized version, but the latter features more complicated formula which only distract attention from the essential point.

We start with (2.4) and substitute the constraint (in this case (A2)). This results in

$$\operatorname{div} \operatorname{grad} p = \operatorname{div} \mathbf{R}. \tag{A3}$$

Next, the equations of motion are solved with a numerical time-integration method. When (3.1) is used, this gives the following time discretization for (A3)

$$\operatorname{div} \operatorname{grad} p^{n+1} = \operatorname{div} \mathbf{R}^n. \tag{A4}$$

Having solved this equation, the velocity at time level $n+1$ is obtained from (3.2), repeated here

$$\mathbf{q}^{n+1} = \mathbf{q}^n + \delta t(\mathbf{R}^n - \operatorname{grad} p^{n+1}). \tag{A5}$$

However, the solution of (A4) cannot be obtained exactly: machine accuracy can be reached at most, and often this equation is solved iteratively until (only) a few figures have converged. Suppose we solve it with an error ε^{n+1}, i.e.,

$$\operatorname{div} \operatorname{grad} p^{n+1} = \operatorname{div} \mathbf{R}^n + \varepsilon^{n+1}.$$

Substitution in (A5) leads to a q^{n+1} whose divergence satisfies

$$\operatorname{div} \mathbf{q}^{n+1} = \operatorname{div} \mathbf{q}^n - \delta t\, \varepsilon^{n+1}. \tag{A6}$$

This is an approximation of the discrete version of (A2) that for the chosen time-integration method would read

$$\frac{1}{\delta t}(\operatorname{div}\mathbf{q}^{n+1}-\operatorname{div}\mathbf{q}^{n})=0.$$

Hence (A2) is satisfied, in a discrete sense, with an error which approaches zero as $\delta t\to 0$.

The same is not true with respect to the original constraint (A1). Error amplification is possible. Suppose a systematic error ε is made each timestep. Then at a fixed time $t=n\,\delta t$ we have

$$\operatorname{div}\mathbf{q}^{n}=-n\,\delta t\,\varepsilon=-t\varepsilon. \tag{A7}$$

Note that it makes no sense letting $\delta t\to 0$. Moreover, when $t\to\infty$, e.g., because we are interested in an equilibrium solution, then we even have $\operatorname{div}\mathbf{q}\to\infty$.

The situation changes when we start with the discretization of the equations of motion. This results in (3.1). A rearrangement of (3.1b) is given in (A5). Only now we demand that the constraint (3.1a) is satisfied, repeated here

$$\operatorname{div}\mathbf{q}^{n+1}=0. \tag{A8}$$

This is a discrete version of the "original" constraint (A1). Combining this with the discrete equations of motion leads to

$$\operatorname{div}\operatorname{grad}p^{n+1}=\frac{1}{\delta t}\operatorname{div}\mathbf{q}^{n}+\operatorname{div}\mathbf{R}^{n}. \tag{A9}$$

This equation has to be compared with (A4). Now when an error ε^{n+1} is made in solving the equation, after substitution in (A5) it results in

$$\operatorname{div}\mathbf{q}^{n+1}=-\delta t\,\varepsilon^{n+1}.$$

In contrast with (A6), here no accumulation of errors is possible. Furthermore, letting $\delta t\to 0$ results in $\operatorname{div}\mathbf{q}^{n}\to 0$.

Remarks. (1) The possibility of error accumulation in solving differential equations with algebraic constraints such as (2.1) was noted two decades ago by Hirt and Harlow [7], and by Gear [8]. The above simple demonstration hereof is not yet widely known.

(2) The situation described in this Appendix is yet another, generalized, application of the philosophy discussed in this paper: the term "boundary condition" has to be replaced by "constraint." The method that is prone to error accumulation combines the analytical equations first, leading to (A3); only thereafter is the time-discretization performed. The other, stable, method first discretizes the equations and combines them with the constraint later.

Acknowledgments. The author is indebted to Mr. J. J. I. M. van Kan of the Delft University of Technology for pointing out the application of the described philosophy to the ψ-ω formulation in § 4, and to Dr. P. Wilders of the Delft University of Technology for pointing out the shallow-water equations application in § 4.

REFERENCES

[1] P. M. Gresho and R. L. Sani, *On pressure boundary conditions for the incompressible Navier-Stokes equations*, Internat. J. Numer. Meth. Fluids, 7 (1987), pp. 1111-1145.

[2] R. PEYRET AND T. D. TAYLOR, *Computational Methods for Fluid Flow*, Springer-Verlag, Berlin, New York, 1983.

[3] R. TEMAM, *Navier–Stokes Equations*, North-Holland, Amsterdam, 1974.

[4] J. E. WELCH, F. H. HARLOW, J. P. SHANNON, AND B. J. DALY, *The* MAC *method, a computing technique for solving viscous incompressible, transient fluid-flow problems involving free surfaces*, Report LA-3425, Los Alamos Research Laboratories, Los Alamos, NM, 1966.

[5] G. I. MARCHUK, *Methods of Numerical Mathematics*, Springer-Verlag, Berlin, New York, 1982.

[6] A. R. MITCHELL AND D. F. GRIFFITHS, *The Finite Difference Method in Partial Differential Equations*, John Wiley, New York, 1980.

[7] C. W. HIRT AND F. H. HARLOW, *A general corrective procedure for the numerical solution of initial-value problems*, J. Comput. Phys., 2 (1967), pp. 114–119.

[8] C. W. GEAR, *Simultaneous numerical solution of differential-algebraic equations*, IEEE Trans. Circuit Theory, 18 (1971), pp. 89–95.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 1, pp. 92-111, January 1990

ADAPTIVE DOMAIN EXTENSION AND ADAPTIVE GRIDS FOR UNBOUNDED SPHERICAL ELLIPTIC PDEs*

M. K. SEAGER† AND G. F. CAREY‡

Abstract. The problem of approximating the solution to a class of (PDEs) posed on unbounded domains using finite domain approximations is considered. A finite-element method is formulated for the approximation on the finite subregions, and a domain extension strategy that "balances" the finite-element error and domain-truncation error is developed. It is shown that this scheme yields asymptotically optimal finite-element approximation properties to the solution on the unbounded domain as the grid is extended. Error estimates for adaptive refinement and domain truncation are developed.

Key words. domain extension, potential equations, optimal grids, finite elements, error estimates

AMS(MOS) subject classifications. 65N30, 65N50

1. Introduction. A class of problems of interest in mathematical physics is characterized by elliptic partial differential equations (PDEs) on unbounded domains. Consequently, the far-field boundary conditions are specified in terms of limits or in terms of concepts such as "bounded at infinity." For example, the irrotational flow of an incompressible fluid exterior to a body in $\mathbb{R}^3$ is described by Laplace's equation in the exterior domain Ω with no flow through the body boundary $\partial\Omega$ and uniform flow at infinity. Similarly, in acoustics and electromagnetism, Hemholtz's equation describes the exterior scattering from a body in $\mathbb{R}^3$. In this case, the "boundary" condition designates the decay far-field rate of the solution at infinity. Elliptic problems on unbounded domains are not only restricted to exterior regions. For instance, there has been recent substantial work in numerical cosmology where the region of interest is all of $\mathbb{R}^3$ [18]. There the problem is to find the conformal transformation from an asymptotically flat Riemann metric into one with prescribed scalar curvature (usually zero). Hence, the problem is posed in $\mathbb{R}^3$ with solution u satisfying $\lim_{r\to\infty} u(x) = 0$ for the transformation factor $u+1$ (e.g., see Kazden and Walker [12]).

Existence and uniqueness of solutions have been obtained by several authors for the cosmology and the Helmholtz scattering problems using various techniques. For the elliptic case, several authors have considered various aspects of the Fredholm alternative (cf. Walker [19] and Lax and Phillips [13]). The most successful techniques evolved from Nirenberg and Walker's work, which utilized polynomially weighted integral estimates (Nirenberg and Walker [14]). Cantor [5] defines weighted Sobolev spaces that provide a framework for elegant existence and uniqueness results. These well-posedness arguments are summarized in Cantor [6]. The requirements on the elliptic operator and data are quite mild; only certain asymptotic (polynomial) decay rates are needed.

Another approach was taken by Babuška [1], who considered an exponentially weighted Sobolev space and the additional assumption that the forcing data f in the differential equation have exponential decay. He showed that the associated bilinear form for the Helmholtz operator satisfies the inf-sup condition. This implies that the

* Received by the editors August 5, 1985; accepted for publication (in revised form) December 6, 1988. This research has been supported in part by the National Science Foundation, the Department of Energy, and Lawrence Livermore National Laboratory, Livermore, California.
† Lawrence Livermore National Laboratory, L-300, P.O. Box 808, Livermore, California 94550.
‡ University of Texas, Austin, Texas 78712.

generalized Lax-Milgram theorem holds, giving existence and uniqueness of the solution in these weighted spaces.

The focus of the present work is the numerical solution of unbounded problems using finite domain truncation. Babuška analyzed finite-element methods for a Poisson type problem, with data f having exponential decay. His method employed uniform finite-element grids of size h on a sequence of finite domains. As the grid spacing, h, is decreased, the support of the approximation grows arbitrarily slowly (i.e., the domain over which the basis elements are nonzero is of size $h^{-\varepsilon}$ for ε arbitrarily small). Hence, the approximating finite-element solution, for fixed h, can be thought of as an approximation to a truncated problem posed with homogeneous Dirichlet ($u=0$) boundary data. Babuška was able to obtain results in the H^1-norm over compact subsets of $\mathbb{R}^3$ showing convergence, in terms of h, at a rate arbitrarily close to optimal. An interesting observation from Babuška's estimates is that the factor of h by which the finite-element solution is suboptimal is exactly the factor by which the grid expanded. No numerical results were given in this paper.

For the exterior scattering problem, several authors have studied numerical approximations to the solution of the Helmholtz equation $\Delta u + k^2 u = 0$ posed on a truncated domain with radiative boundary condition (cf. Fix and Marin [9], Engquist and Majda [8], Bayliss, Gunzberger, and Turkel [3], and Goldstein [10]). The first-order radiative boundary condition ($\partial u/\partial r - iku = 0$) in these works was first justified by heuristic arguments and later proven by asymptotic expansion methods based on the wave-like properties of solutions. Bayliss also gave asymptotic forms for higher-order boundary (radiative-like) conditions and proposed boundary conditions of order greater than two [4]. The asymptotic estimates for the accuracy of radiative-like boundary conditions rely heavily on the assumption that $k \in \mathbb{R}$ is nonzero. In fact, their estimates diverge as k vanishes. With the exception of Goldstein's method, the above numerical techniques were restricted to uniform grids.

Goldstein's algorithm for the truncated Helmholtz equation with radiative boundary conditions appears to be the first step toward the use of nonuniform grids with domain extension for unbounded problems. He extended this work to Laplace's equation in $\mathbb{R}^3$ with a Robin (convective or mixed) boundary condition. The extension to Laplace's equation required different arguments, based on the assumption of compact support for the forcing function, in order to prove convergence of the numerical algorithm [11].

Cantor [6] extended his weighted Sobolev treatment on all of $\mathbb{R}^n (n \geqq 3)$ to a theoretical approximation scheme based on using truncated domain and boundary conditions of varying degree at the artificial radius $r = R$. Three types of boundary conditions were considered: (1) Dirichlet ($u=0$); (2) Robin ($R\,\partial u/\partial r + u = 0$); and (3) the second-order boundary conditions

$$\frac{R^2}{(n-1)(n-2)} u_{rr} + \frac{2R}{(n-2)} u_r + u = 0.$$

In that study, the error resulting from truncating the domain was studied in an asymptotic sense. For data with polynomial decay rates at infinity, asymptotic convergence estimates were proven (in terms of the artificial radius) in L^2 and H^1 norms over fixed and expanding regions. Improved rates of convergence were found for both Robin and second-order boundary conditions. In the sense that higher-order boundary conditions are used, these ideas are similar to the (radiative-like) boundary conditions applied to the wave equation.

The present study is devoted to the development of an adaptive finite-element approximation scheme using the truncated domain and higher-order boundary condition approach of Cantor [7]. The basic theme is to adaptively extend a bounded domain and adaptively refine to balance domain and mesh errors. The grid is adaptively refined, using the ideas of Babuška and Rheinboldt [2] modified to spherically symmetric norms. A posteriori energy estimators in these norms are devised and used to guide the mesh refinement. A robust domain extension algorithm is devised.

2. Weighted Sobolev spaces and domain extension. Let us consider the boundary value problem posed on an unbounded domain (UBVP):

$$-\nabla \cdot (A\nabla u) + cu = f \quad \text{on } \mathbb{R}^n, \tag{1}$$

$$u(x) \to 0 \quad \text{as } \|x\| \to \infty \tag{2}$$

and the solution, u_R, to the boundary value problem posed on a bounded domain (BBVP):

$$-\nabla \cdot (A\nabla u_R) + cu_R = f \quad \text{on } \Omega_R = B(0, R), \tag{3}$$

$$\beta_1(u_R) = u_R + \frac{(n-2)}{R}\left(\frac{\partial u_R}{\partial \nu_A}\right) = 0 \quad \text{on } \Gamma_R = \partial\Omega_R \tag{4}$$

where $\partial u/\partial \nu_A = (A \cdot \nabla u) \cdot \nu$ is the conormal associated with A, and $B(\alpha, \beta) = \{x \in R^n \mid \|x - \alpha\| < \beta\}$; A is an $n \times n$ matrix function of position x, and c, f are scalar functions of x. Further restrictions on A, c, and f will be given subsequently.

Although the truncated domain analysis will concentrate on the (first-order) asymptotic boundary condition $\beta_1(u_R) = 0$, the Dirichlet (or zero-order), and second-order boundary conditions may also be applied and are defined, respectively, by

$$\beta_0(u_R) = u_R = 0 \quad \text{on } \Gamma_R, \tag{5}$$

$$\beta_2(u_R) = u_R + \frac{2R}{n-2}\frac{\partial u_R}{\partial r} + \frac{R^2}{(n-1)(n-2)}\frac{\partial^2 u_R}{\partial r^2} = 0 \quad \text{on } \Gamma_R. \tag{6}$$

As noted by Bayliss [3], [4], for the Helmholtz equation, $\beta_k(u_R)$ annihilates the first k terms in the asymptotic expansion for u.

The pertinent weighted Sobolev spaces to be used here were first defined by Cantor (see [6] and the references therein for details). The weight function, $\sigma: \mathbb{R}^n \to \mathbb{R}$, is defined pointwise by

$$\sigma(x) = (1 + \|x\|^2)^{1/2}.$$

For $s \in N$, $1 \leqq p \leqq \infty$, and $\delta \in \mathbb{R}$ we define the weighted Sobolev spaces $M^p_{s,\delta}(\mathbb{R}^n)$ as the completion of $C_0^\infty(\mathbb{R}^n)$ with respect to the norm

$$\|u\|_{p,s,\delta} = \left(\sum_{|\alpha| \leqq s} \|\sigma^{\delta+|\alpha|} D^\alpha u\|_p^p\right)^{1/p}, \tag{7}$$

where $\|\cdot\|_p$ is the standard $L^p(\mathbb{R}^n)$ norm and D^α is the usual multi-index derivative. The functions in $M^p_{s,\delta}$ have interesting decay properties, which are used in the subsequent error analysis.

Another norm that will be of interest is the energy norm

$$(u, v)_{E,\Omega} = \int_\Omega (ADu \cdot Dv + cuv)\, dx, \qquad \|u\|_{E,\Omega} = [(u, u)_{E,\Omega}]^{1/2},$$

where A and c are functions defining the boundary value problem and Ω is some (measurable) subset of the domain of A and f.

For the actual implementation of ideas developed in this study, we will need function spaces based on spherically symmetric norms. These norms and spaces are defined in the same way as the standard Hilbert spaces, viz.,

$$(8)\qquad \|v\|_{S,0,(R_0,R)}=\left(\int_{R_0}^{R} v^2r^2\,dr\right)^{1/2},\qquad \|v\|_{S,m,(R_0,R)}=\left(\sum_{|\alpha|\leqq m}\|D^\alpha v\|^2_{S,0,(R_0,R)}\right)^{1/2}$$

where $H^m_S(R_0,R)(H^m_{0S}(R_0,R)$, respectively) is defined to be the completion of $C^\infty(R_0,R)(C_0^\infty(R_0,R)$, respectively) with respect to the norms in (8). In the above, $0\leqq R_0<R$ and m is a nonnegative integer. It is easily seen that, under appropriate conditions on the boundary value problem, the standard Sobolev norm on W_p^m and (8) are equivalent norms over $H_S^m(R_0,R)$ and that the constants in the equivalence relationships depend on R_0 and R. Many properties of functions in $M^p_{s,\delta}$ have been shown by Cantor [6], [7]. We now use these results to derive asymptotic error estimates in terms of the truncation radius R of the domain.

THEOREM 1. *Let* $e_R=u-u_R$, $p>1$, $s>n/p+2$, $0\leqq\beta<1$, *and* $\delta=n/q-2-(n-\beta-2)/2$ *hold. Let A be uniformly elliptic such that* $(a_{ij}-\delta_{ij})\in M^p_{s,\delta}$. *Suppose* $c\geqq 0$ *with* $c\in M^p_{s-1,\delta-2}$ *and* $f\in M^p_{s-2,n/q+\beta}$. *Then, if u_R is the solution to* (3), *we have*

$$(9)\qquad \|e_R\|_{2,\Omega_R}=o(R^{-\alpha})\qquad \forall\alpha<\beta-(4-n)/2,$$

$$(10)\qquad \|\nabla e_R\|_{2,\Omega_R}=o(R^{-\gamma})\qquad \forall\gamma<\beta+(n-2)/2,$$

$$(11)\qquad \|e_R\|_{E,\Omega_R}=o(R^{-\gamma})\qquad \forall\gamma<\beta+(n-2)/2.$$

Proof. For the proof of (9), we follow Cantor [7] and first determine the asymptotic behavior of $\partial u/\partial\nu_A$. To this end, let $r=\sum_{i=1}^n r_ie_i$ be the radial vector. Then $\partial u/\partial\nu_A-\partial u/\partial\nu=\sum_{i,j=1}^n(a_{ij}-\delta_{ij})r_i\,\partial u/\partial x_j$. By assumption $(a_{ij}-\delta_{ij})\in M^p_{s,\delta}$. Also, from the definition of $M^p_{s,\delta}$, $\partial u/\partial x_j\in M^p_{s-1,\delta+1}$, which implies $(\partial u/\partial\nu_A-\partial u/\partial\nu)\in M^p_{s-1,l}$ for all $l<2\delta+1+n/p$. Thus, by Cantor [6, Thm. 5.4]

$$(12)\qquad \left(\frac{\partial u}{\partial\nu_A}-\frac{\partial u}{\partial\nu}\right)\sigma^\gamma\to 0$$

for all γ such that

$$\gamma<2\delta+2n/p+1=2[n/q-2-(n-\beta-2)/2]+1+\frac{2n}{p}$$
$$=n-1+\beta.$$

By applying Cantor [7, Lem. 11] and (2), we have

$$\left\|\frac{\partial u}{\partial\nu_A}-\frac{\partial u}{\partial\nu}\right\|^2_{2,\Gamma_R}=\int_{\Gamma_R}\left(\frac{\partial u}{\partial\nu_A}-\frac{\partial u}{\partial\nu}\right)^2R^{n-1}\,ds$$
$$\leqq C\int_{\Gamma_R}\sigma^{-2\gamma+n-1}(R)\,ds\quad\text{as } R\to\infty$$
$$\leqq C\sigma^{-2\gamma+n-2}\qquad\text{as } R\to\infty$$

and, hence,

$$\lim_{R\to\infty}\left\|\frac{\partial u}{\partial\nu_A}-\frac{\partial u}{\partial\nu}\right\|^2_{2,\Gamma_R}\sigma^{2t}=0$$

for

$$t<(2\gamma-n+2)/2<[2(n-1)+2\beta-n+1]/2<\frac{n-1}{2}+\beta.$$

Now that the decay rate of the conormal is established, we turn to the $L^2(\Omega_R)$ estimate of e_R. From Cantor [7, Lem. 11], we have

$$C_1R^2\|\nabla e_R\|^2_{2,\Omega_R}+\|e_R\|^2_{2,\Omega_R}\leqq C_2R\|\beta_1(u)\|^2_{2,\Gamma_R}. \tag{13}$$

This implies

$$\begin{aligned}\|e_R\|_{2,\Omega_R}&\leqq CR^{1/2}\|\beta_1(u)\|_{2,\Gamma_R}\\ &\leqq CR^{1/2}\left(\left\|u+\frac{R}{n-2}\frac{\partial u}{\partial\nu}\right\|_{2,\Gamma_R}+\frac{R}{n-2}\left\|\frac{\partial u}{\partial\nu_A}-\frac{\partial u}{\partial\nu}\right\|_{2,\Gamma_R}\right).\end{aligned}$$

The second term has been analyzed earlier. To obtain the decay rate of the first term, we note that $[u+R(\partial u/\partial\nu)/(n-2)]\sigma^\gamma(R)\to 0$ for $\gamma<n-2+\beta$. An argument similar to the one above gives

$$\lim_{R\to\infty}\left\|u+\frac{R}{n-2}\frac{\partial u}{\partial\nu}\right\|_{2,\Gamma_R}\sigma^t=0\quad\forall t<(n-3)/2+\beta.$$

Now, let $\alpha<\beta-(4-n)/2$ hold. Then

$$\|e_R\|_{2,\Omega_R}R^\alpha\leqq C\left(\left\|u+\frac{R}{n-2}\frac{\partial u}{\partial\nu}\right\|_{2,\Gamma_R}R^{\alpha+1/2}+\left\|\frac{\partial u}{\partial\nu_A}-\frac{\partial u}{\partial\nu}\right\|_{2,\Gamma_R}R^{\alpha+3/2}\right).$$

But $\alpha+1/2<\beta-(3-n)/2$ and $\alpha+3/2<\beta-(1-n)/2$ so that

$$\lim_{R\to\infty}\|e_R\|_{2,\Omega_R}R^\alpha=0\quad\forall\alpha<\beta-(4-n)/2.$$

For the estimate on $\|\nabla e_R\|_{2,\Omega_R}$ we return to inequality (13) and neglect the second term to obtain

$$\|\nabla e_R\|_{2,\Omega_R}\leqq CR^{-1/2}\|\beta_1(u)\|^2_{2,\Gamma_R}.$$

Next let $\alpha<\beta+(n-2)/2$ hold. Then

$$\|\nabla e_R\|_{2,\Omega_R}R^\alpha\leqq C_1R^{\alpha-1/2}\left\|u+\frac{R}{n-2}\frac{\partial u}{\partial\nu}\right\|_{2,\Gamma_R}+C_2R^{\alpha+1/2}\left\|\frac{\partial u}{\partial\nu_A}-\frac{\partial u}{\partial\nu}\right\|_{2,\Gamma_R}.$$

Using $\alpha-1/2<\beta+(n-3)/2$ and $\alpha+1/2<\beta+(n-1)/2$,

$$\lim_{R\to\infty}\|\nabla e_R\|_{2,\Omega_R}R^\alpha=0\quad\forall\alpha<\beta+(n-2)/2.$$

Finally, to obtain (11), consider the energy norm defined by the associated bilinear form

$$\|e_R\|^2_{E,\Omega_R}=\int_{\Omega_R}(A\nabla e_R\cdot\nabla e_R+ce_R^2)\,dx+\frac{(n-2)}{R}\int_{\Gamma_R}e_R^2\,ds. \tag{14}$$

To bound the boundary integral on the right-hand side of (14), we use the assumptions that A be uniformly elliptic and $c\geqq 0$. Then

$$\frac{(n-2)}{R}\int_{\Gamma_R}e_R^2\,ds\leqq\frac{(n-2)}{R}\|\beta_1(u)\|_{2,\Gamma_R}\|e_R\|_{2,\Gamma_R}$$

and so

$$\frac{(n-2)}{R}\int_{\Gamma_R}e_R^2\,ds\leqq\frac{(n-2)}{R}\|\beta_1(u)\|^2_{2,\Gamma_R}.$$

From the proof of (9), we have

$$\lim_{R\to\infty} \|\beta_1(u)\|^2_{2,\Gamma_R} R^\alpha = 0 \quad \forall \alpha < 2\beta + n - 3. \tag{15}$$

Hence

$$\lim_{R\to\infty} \frac{(n-2)}{R} \int_{\Gamma_R} e_R^2 \, ds \, R^{2\gamma} = 0 \quad \forall 2\gamma < 2\beta + n - 2. \tag{16}$$

To estimate the first-order term in (14), we use (9) and (15) to obtain

$$\begin{aligned} 0 &\leqq \lim_{R\to\infty} \left[\int_{\Omega_R} (A\nabla e_R \cdot \nabla e_R) \, dx \, R^{2\gamma} + \frac{(\varepsilon-1)(n-2)}{\varepsilon} \|e_R\|^2_{2,\Gamma_R} R^{2\gamma-1} \right] \\ &\leqq \lim_{R\to\infty} \|\beta_1(u)\|^2_{2,\Gamma_R} R^{2\gamma-1} = 0 \quad \forall 2\gamma < 2\beta + n - 2. \end{aligned} \tag{17}$$

We now can estimate the remaining term in the expression for the energy norm. To this end, recall $c \in M^p_{s-2,\delta+2}$. Hence,

$$\|c\sigma^t\|_{\infty,R^n} < C\|c\|_{p,s-2,\delta+2} \quad \forall t < n/p + \delta + 2.$$

From the definition of δ this is equivalent to

$$|c(x)| \leqq C\|c\|_{2,s-2,\delta+2}\sigma^{-t} \quad \forall t < (n-2+\beta)/2.$$

Thus, we can write

$$R^{2\gamma} \int_{\Omega_R} c e_R^2 \, dx \leqq C\|c\|_{2,s-2,\delta+2} R^{2\gamma-t} \int_{\Omega_R} e_R^2 \, dx \tag{18}$$

which, from (9), goes to zero as $R \to \infty$ for all

$$2\gamma - t < 2\beta - (4-n)$$

or

$$\begin{aligned} 2\gamma &< 2\beta - 4 + n + (n+2+\beta)/2 \\ &= 2\beta + (n-2) + [(n+\beta)/2 - 1]. \end{aligned}$$

This will hold (since $n \geqq 3$) if we require $2\gamma < 2\beta + (n-2)$.

Combining this result with (16) and (17), we see that

$$\lim_{R\to\infty} R^\gamma \|e_R\|_{E,\Omega_R} = 0 \quad \forall \gamma < \beta + (n-2)/2. \qquad \square$$

As an example, in the case $n = 3$, we have

$$\begin{aligned} \|e_R\|_{2,\Omega} &= o(R^{-\alpha}) \quad && \forall \alpha < 1/2, \\ \|\nabla e_R\|_{2,\Omega} &= o(R^{-\gamma}) && \forall \gamma < 3/2, \\ \|e_R\|_{E,\Omega} &= o(R^{-\gamma}) && \forall \gamma < 3/2, \end{aligned}$$

a rather slow decay rate. We must keep in mind, however, that dV (in the spherically symmetric case) increases like R^2. This is going to amplify any small errors in the asymptotic region. It is interesting to inquire what would happen if we set $R_N < R$ fixed and examined the errors over the subdomain Ω_{R_N} alone as the problem domain Ω_R increases. We anticipate that the asymptotic rates will be better than those in Theorem 1 for Ω_R.

THEOREM 2. *Let e_R be defined as in Theorem* 1 *and R_N fixed, with $0 < R_N < R$; then*

$$\|e_R\|_{2,\Omega_{R_N}} = o(R^{-\sigma}), \quad \|e_R\|_{1,2,\Omega_{R_N}} = o(R^{-\alpha}) \quad \forall \alpha < \beta + (n-2)/2. \tag{19}$$

Proof. See Cantor [7]. □

Now that estimates for the domain-truncation error have been derived in several norms, we turn to approximating u_R, the solution of the BBVP. For the following, we will restrict our attention to the spherically symmetric case $n = 3$ and the scalar operator $A = a$ with $a = a(x) > 0$ in the domain. Writing the weak formulation of the BBVP (3)-(4) in terms of both the artificial radius R and the (possibly) exterior radius R_0, we have

$$\begin{aligned} B(u_R, v) &= \int_{R_0}^{R} \left(a \frac{\partial u_R}{\partial r} \frac{\partial v}{\partial r} + c u_R v \right) r^2 \, dr + R_0^2 u_0' v(R_0) + R u_R(R) v(R) \\ &= \int_{R_0}^{R} f v r^2 \, dr = f(v). \end{aligned}$$

Note that the above boundary data at $r = R$ implies the use of the Robin boundary condition. Similarly, the boundary data at $r = R_0$ implies the use of flux boundary condition ($u_R'(R_0) = u_0'$, given).

For $R_0 > 0$, we consider a standard C^0-conforming finite-element approximation. To this end, let a grid with $G+1$ knots $\{R_0 = x_0 < x_1 < \cdots < x_G = R\}$ be given. Define the local mesh parameter $h_j = x_j - x_{j-1}$ to be the length of element $\Omega_j = (x_{j-1}, x_j)$, $j = 1, 2, \cdots, G$. Let $\{\phi_j\}_{j=1}^{n}$ be the global finite-element basis. (If $R_0 = 0$, a finite-element basis can be constructed that handles the singularity in the spherical Laplacian in a natural way and has the standard approximation properties [15], [16].) In the following treatment we take $R_0 \neq 0$ and assume all elements are of degree k. Later, in § 3 the particular case of linear elements is examined to obtain error indicators for adaptive mesh refinement. Then, for the domain extension we return to the case of degree $k \geqq 1$.

The approximation space is $S^h = \operatorname{span}\{\phi_j\} \subset H_S^1(R_0, R)$. Then the finite-element approximation u_h satisfies

$$B(u_h, \phi) = f(\phi) \quad \forall \phi \in S^h.$$

The following theorem decomposes the error induced by numerically approximating the UBVP into its constituent parts.

THEOREM 3. *Let the conditions of Theorem* 1 *hold for all $\beta < 1$. Then $\gamma < \frac{3}{2}$,*

$$\|u - u_h\|_{SE,(R_0,R)} = o(R^{-\gamma}) + O(h^k) \tag{20}$$

for $R \to \infty$, $h \to 0$, where k is the degree of the finite-element basis and $\|\cdot\|_{SE}$ is the spherical energy norm. (That is, the energy norm in spherical form based on (8).)

Proof. The proof follows as a straightforward application of the triangle inequality together with Cea's lemma, Theorem 1, and the finite-element interpolation property. □

Since two parameters h and R enter the result, this is not satisfactory from a practical standpoint. To analyze this estimate further, we are led to consider relating h and R. If $R = R(h)$ is known, then given h, we could determine a right-hand endpoint, a priori. From (20), we see that to keep the domain-truncation error and discretization error of the same order we seek a relation of the form

$$R^{-\gamma} = h^k$$

or

$$R = h^{-k/\gamma}$$

where $\gamma < \frac{3}{2}$. To interpret this result, let $\gamma = 1.49$ and $k = 2$ (quadratics). Then $R = h^{-1.3423}$ and for $h = 0.1$, 0.01, and 0.001, we obtain $R = 21.99$, 483.69, and 10,637, respectively.

Note that here $h = \max(h_1, h_2, \cdots, h_G)$ and in practice is not the asymptotic parameter of interest; rather, we seek estimates in terms of

$$H = DIM(S^h)^{-1} = N^{-1}$$

with $C_1 R h^{-1} \leqq N \leqq C_2 R h^{-1}$, constants C_1, C_2, independent of R. Then

$$C_1 h^k \leqq H^\rho \leqq C_2 h^k,$$

where $\rho = \gamma k/(k+\gamma)$. Thus, we have the following theorem.

THEOREM 4. *Let the conditions of Theorem* 1 *hold for all* $\beta < 1$. *For* $H = N^{-1}$, $R = h^{-k/\gamma}$,

$$\|u - u_h\|_{SE,(R_0,R)} = O(H^{\gamma k/(k+\gamma)}) \quad \forall \gamma < \tfrac{3}{2}. \tag{21}$$ □

For $\gamma = \frac{3}{2}$, this is the same result as in Goldstein [11], who notes that this estimate is suboptimal (optimal being H^k). In order to obtain an optimal result, we need to consider a nonuniform mesh in which the knot spacing increases (with polynomial growth) as $R \to \infty$. That is, a more appropriate grid must be constructed. Clearly, this grid should take advantage of the asymptotic behavior of u. The next two sections give the preliminaries for defining such a grid and, in the process, determine an accurate estimator of the finite-element energy error.

3. Spherically symmetric asymptotically optimal grids.

We now seek to classify an asymptotically optimal grid. To this end we consider the error indicators developed in Seager and Carey [17] and extend them so as to classify an optimal grid. The analysis of error indicators here is restricted to linear elements. We begin by considering a projection operator

$$P : H_S^1(R_0, R) \to \{u \in H_S^1(R_0, R) \mid u(x_j) = 0, j = 1, 2, \cdots, G\}$$

with respect to the scalar product defined by $B(u, v)$. Let u_h be the piecewise-linear ($k = 1$) finite-element solution on (R_0, R), and $e = u_R - u_h$. The following analysis uses the error projection Pe to relate the energy norm of the error to a "mesh transformation."

Now, Pe satisfies $L(Pe) = Lu_h - f$ on $\Omega_e = (x_{j-1}, x_j)$ with $Pe(x_{j-1}) = Pe(x_j) = 0$. Let z_e denote $Pe|_{\Omega_e}$. Then the corresponding weak statement is: Find $z_e \in H_0^1(\Omega_e)$ such that

$$(z_e, v)_{SE,\Omega_e} = (L[u_h] - f, v)_{S,\Omega_e}, \qquad e = 1, 2, 3, \cdots, G$$

for all $v \in H_0^1(\Omega_e)$. Also, we have

$$\|Pe\|_{SE}^2 = \sum_{e=1}^{G} \|z_e\|_{SE,\Omega_e}^2. \tag{22}$$

The values $\|z_e\|_{SE,\Omega_e}^2$ are related to the error indicators ε_e^2 [17].

We now transform z_e into a quantity more readily useful for characterizing the optimum grid.

Because u_h is linear on Ω_e,

$$\begin{aligned} L[u_h] - f &= -\frac{1}{r^2}(r^2 a u_h')' + c u_h - f \\ &= -\left(\frac{2}{r} a + a'\right) u_h' + c u_h - f \\ &= \rho_e + \tau_e \end{aligned}$$

where

$$\rho_e(r) = a(r)u_R''(r), \qquad \tau_e(r) = -\left(\frac{2}{r}a + a'\right)e'(r) + ce(r).$$

Next let $\phi_e, \psi_e \in H^1_{0,S}(\Omega_e)$ be such that

$$(\phi_e, v)_{SE,\Omega_e} = (\rho_e, v)_{S,\Omega_e} \tag{23}$$

$$(\psi_e, v)_{SE,\Omega_e} = (\tau_e, v)_{S,\Omega_e} \tag{24}$$

for all $v \in H^1_{0,S}(\Omega_e)$. Thus, z_e is decomposed into $\phi_e + \psi_e$. From the above definitions of ρ_e and τ_e and the fact that the weak projection of Pe on Ω_e is well defined, ϕ_e should be the dominant contribution to $z_e = Pe|_{\Omega_e}$. This is the case, and we begin by showing that ψ_e gives a higher-order contribution to the energy norm of z_e. Recall that the smallest eigenvalue of L on Ω_e with zero Dirichlet boundary conditions is bounded below by the smallest eigenvalue $a_{\min,e}(\pi/h_e)^2$ of the operator $-a_{\min}(1/r^2)\, d/dr(r^2(d/dr))$ on Ω_e, where

$$a_{\min,e} = \min_{x_{e-1} \leqq r \leqq x_e} |a(r)|,$$

$$a_{\min} = \min_{R_0 \leqq r \leqq R} |a(r)|.$$

Hence, (24) implies

$$\begin{aligned} Ch_e^{-2}\|\psi_e\|^2_{S,\Omega_e} &= Ch_e^{-2}(\psi_e, \psi_e)_{S,\Omega_e} \leqq (L[\psi_e], \psi_e)_{S,\Omega_e} \\ &= (\psi_e, \psi_e)_{SE,\Omega_e} = (\tau_e, \psi_e)_{S,\Omega_e} \\ &\leqq \|\tau_e\|_{S,\Omega_e}\|\psi_e\|_{S,\Omega_e} \end{aligned}$$

or

$$\|\psi_e\|_{S,\Omega_e} \leqq Ch_e^2\|\tau_e\|_{S,\Omega_e}.$$

Therefore,

$$\|\psi_e\|^2_{SE,\Omega_e} = (\tau_e, \psi_e)_{S,\Omega_e} \leqq \|\tau_e\|_{S,\Omega_e}\|\psi_e\|_{S,\Omega_e} < Ch_e^2\|\tau_e\|^2_{S,\Omega_e}. \tag{25}$$

On the other hand,

$$\begin{aligned} \|\tau_e\|^2_{S,\Omega_e} &= \int_{\Omega_e}\left[-\left(\frac{2}{r}a + a'\right)e' + ce\right]^2 r^2\,dr \\ &\leqq \int_{\Omega_e} 2\left[\left(\frac{2}{r}a + a'\right)^2(e')^2 + c^2e^2\right] r^2\,dr \\ &\leqq \frac{C}{R_0^2}\|e\|^2_{SE,\Omega_e} \end{aligned}$$

which together with (25) yields

$$\|\psi_e\|_{SE,\Omega_e} \leqq \frac{C}{R_0} h_e \|e\|_{SE,\Omega_e}. \tag{26}$$

Next introduce the quantities

$$S = \frac{1}{\|e\|_{SE}}\left[\sum_{e=1}^{G}\|\psi_e\|^2_{SE,\Omega_e}\right]^{1/2}$$

and

$$Q^2=\sum_{e=1}^{G}\|\varphi_e\|^2_{SE,\Omega_e}. \tag{27}$$

It can be shown (Seager and Carey [17]) that Pe is equivalent to e up to $O(h)$ in the spherical energy norm. That is,

$$\|Pe\|_{SE}\leqq\|e\|_{SE}\leqq\|Pe\|_{SE}(1+O(h)).$$

Together with (22), this yields

$$\begin{aligned}\|e\|^2_{SE}&=\sum_{e=1}^{G}\|z_e\|^2_{SE,\Omega_e}[1+O(h)]\\&=\sum_{e=1}^{G}(\varphi_e+\psi_e,\varphi_e+\psi_e)_{SE,\Omega_e}[1+O(h)]\\&=[Q^2+2\alpha QS\|e\|_{SE,(R_0,R)}+S^2\|e\|^2_{SE,(R_0,R)}][1+O(h)]\\&=[Q+\alpha S\|e\|_{SE,(R_0,R)}]^2[1+O(h)]+(1-\alpha^2)S^2\|e\|^2_{SE,(R_0,R)}[1+O(h)]\end{aligned}$$

where $|\alpha|\leqq1$. By (26) we have $S=O(h)$, and hence

$$\|e\|^2_{SE,(R_0,R)}=(Q+\alpha S\|e\|_{SE,(R_0,R)})^2[1+O(h)] \tag{28}$$

which yields

$$\|e\|_{SE,(R_0,R)}=Q[1+O(h)]. \tag{29}$$

Now let us characterize the quantity Q.

LEMMA 1. *Suppose that $u''_R\neq0$ on (R_0,R). Set $\rho_{\max,e}=\max\{|\rho(r)|;\ r\in\Omega_e\}$, and $\bar{x}_e$ as the element midpoint. Then*

$$Q^2=\left[\frac{1}{12}\sum_{e=1}^{G}\frac{\rho^2_{\max,e}}{a(\bar{x}_e)}h_e^3\bar{x}_e^2\right][1+O(h)]. \tag{30}$$

Proof. Set

$$\sigma_e(r)=\rho(r)-\rho_{\max,e}\quad\text{on }\Omega_e \tag{31}$$

and split ϕ_e into $\phi_{1e}+\phi_{2e}$ by defining $\phi_{1e},\phi_{2e}\in H_0^1(\Omega_e)$ such that

$$(\varphi_{1e},v)_{E,\Omega_e}=(\rho_{\max,e},v)_{\Omega_e},\qquad(\varphi_{2e},v)_{E,\Omega_e}=(\sigma_e,v)_{\Omega_e} \tag{32}$$

for all $v\in H_0^1(\Omega_e)$.

By assumption, there exists $\rho_0>0$ such that $|\rho(x)|\geqq\rho_0$ on (R_0,R). Hence, for small h,

$$|\sigma_e(x)|\leqq C\frac{\rho_{\max,e}}{\rho_0}h_e. \tag{33}$$

Recall that on Ω_e

$$a_{\min,e}=\min\{a(x)\,|\,x\in\Omega_e\}=a(\bar{x}_e)[1+O(h)]. \tag{34}$$

Since for all $v\in H_0^1(\Omega_e)$ the Sobolev inequality gives

$$\|v\|^2_{E,\Omega_e}=\int_{\Omega_e}a(r)[v'(r)]^2\,dr[1+O(h^2)]$$

it follows from (31) and (32) and the mean value theorem for integrals that

$$\begin{aligned}\|\varphi_{1e}\|^2_{SE,\Omega_e} &= \gamma_e^2\|\varphi_{1e}\|^2_{E,\Omega_e} = \gamma_e^2\left[\sup_{v\in H_0^1(\Omega_e)} \frac{|(\rho_{\max,e}, v)_{\Omega_e}|}{\|v\|_{E,\Omega_e}}\right]^2 \\ &= \gamma_r^2 \frac{\rho^2_{\max,e}}{a(\bar{x}_e)} \sup_{v\in H_0^1(\Omega_e)} \frac{|\int_{\Omega_e} v\,dr|^2}{\int_{\Omega_e}(v')^2\,dr}[1+O(h)] \\ &= \gamma_e^2 \frac{\rho^2_{\max,e}}{a(\bar{x}_e)} \int_{\Omega_e} (\tilde{v}')^2\,dr[1+O(h)]\end{aligned}$$

where $\gamma_e\in[x_{e-1}, x_e]$ and $\tilde{v}\in H_0^1(\Omega_e)$ such that $-\tilde{v}''=1$. This implies that

$$\|\varphi_{1e}\|^2_{SE,\Omega_e} = \frac{1}{12}\frac{\rho^2_{\max,e}}{a(\bar{x}_e)}\gamma_e^2 h_e^3[1+O(h)]. \tag{35}$$

To estimate ϕ_{2e} we proceed as follows: The smallest eigenvalue of $-d/dx[a(x)\,d/dx]$ on Ω_e is bounded below by $a_{\min,e}(\pi/h_e)^2$. Hence, by (31) and the mean value theorem for integrals

$$\begin{aligned}\|\varphi_{2e}\|^2_{SE,\Omega_e} &= \tilde{\gamma}_e^2\|\varphi_{2e}\|^2_{E,\Omega_e} = \tilde{\gamma}_e^2(\sigma_e, \varphi_{2e})_{\Omega_e} \\ &\leqq \tilde{\gamma}_e^2\|\varphi_{2e}\|_{\Omega_e}\|\sigma_e\|_{\Omega_e} \\ &\leqq \frac{1}{a_{\min,e}}\left(\frac{h_e\tilde{\gamma}_e}{\pi}\right)^2\|\sigma_e\|^2_{\Omega_e} \\ &\leqq \frac{C}{a(\bar{x}_e)}\left(\frac{\gamma_e\rho_{\max,e}}{\pi}\right)^2 h_e^5[1+O(h)]\end{aligned}$$

where $\tilde{\gamma}_e\in[x_{e-1}, x_e]$ and $|\gamma_e-\tilde{\gamma}_e|=O(h)$. Combining this with (35) we obtain, for some $|\alpha|\leqq 1$,

$$\begin{aligned}\|\varphi_e\|^2_{SE,\Omega_e} &= \|\varphi_{1e}\|^2_{SE,\Omega_e} + 2\alpha\|\varphi_{1e}\|_{SE,\Omega_e}\|\varphi_{2e}\|_{SE,\Omega_e} + \|\varphi_{2e}\|^2_{SE,\Omega_e} \\ &= \frac{1}{12}\frac{\rho^2_{\max,e}}{a(\bar{x}_e)} h_e^3\gamma_e^2[1+O(h)].\end{aligned}$$

The result (30) then follows from the definition of Q^2. □

Let us now characterize a class of (possibly nonuniform) partitions. A partition Δ is an (ξ, G)-partition if there exists a sufficiently smooth piecewise map ξ from $[R_0, R]$ onto $[0, 1]$ that maps Δ onto a uniform partition with $G+1$ knots on $[0, 1]$. More precisely, Δ is an (ξ, G)-partition if, for some function $\xi:[R_0, R]\to[0, 1]$,

$$\xi(x_j) = j/G, \qquad j=0, 1, 2, \cdots, G$$

with

$$\xi\in\{v\in H^1(R_0, R)|\quad |v|_{\Omega_e^*}\in C^2(\Omega_e^*), v'(x)\geqq\delta>0 \text{ on each } \Omega_e^*, v(R_0)=0, v(R)=1\}$$

where

$$\Omega_e^* = \left(\frac{(e-1)(R-R_0)}{G}+R_0, \frac{e(R-R_0)}{G}+R_0\right).$$

Note that for an (ξ, G)-partition we have

$$\frac{1}{G} = \int_{\Omega_e} \xi'(t)\,dt = h_e\xi'(\bar{x}_e)[1+O(h)]. \tag{36}$$

With this established, we can characterize the spherical energy norm of the error in terms of the true solution and the (ξ, G)-partition Δ.

THEOREM 5. *For the (ξ, G)-partition Δ the error satisfies*

$$\|e\|^2_{SE} = \frac{1}{12G^2}\int_{R_0}^{R}\left[\frac{\rho(r)}{\xi'(r)}\right]^2\frac{r^2}{a(r)}\,dr[1+O(h)]. \tag{37}$$

Proof. Because $\rho \in C^1(R_0, R)$ and $1/\xi'$ is piecewise $C^1(R_0, R)$, the Riemann sum (30) for Q^2 can be written as an integral for small h. Hence, using (36)

$$Q^2 = \frac{1}{12G^2} \int_{R_0}^{R} \left| \frac{\rho(r)}{\xi'(r)} \right|^2 \frac{r^2}{a(r)} dr[1 + O(h)].$$

The theorem now follows from (29). □

Since the error norm (37) depends on ξ, in order to minimize the error it is reasonable to minimize the variational functional

$$J(u) = \int_{R_0}^{R} \left[\frac{\rho(r)}{\xi'(r)} \right]^2 \frac{r^2}{a(r)} dr \tag{38}$$

with respect to ξ subject to the conditions

$$\xi(R_0) = 0, \qquad \xi(R) = 1.$$

The resulting Euler–Lagrange equation is

$$\frac{d}{dr} \left\{ \frac{\rho^2(r)}{[\xi'(r)]^3} \frac{r^2}{a(r)} \right\} = 0,$$

and is directly solvable. Thus, we have proven the following theorem.

THEOREM 6. *For the spherically symmetric form of boundary value problem* (1) *on* $[R_0, R]$, *the grid that minimizes the spherical energy norm* (*given a fixed number of degrees of freedom*) *is given by the transformation* $\xi_0 : [R_0, R] \to [0, 1]$,

$$\xi_0(r) = \gamma_0 \int_{R_0}^{r} \left[\frac{\rho^2(t)}{a(t)} t^2 \right]^{1/3} dt \tag{39}$$

where

$$\gamma_0^{-1} = \int_{R_0}^{R} \left[\frac{\rho^2(t)}{a(t)} t^2 \right]^{1/3} dt.$$

The actual grid $\{x_0, x_1, \cdots, x_G\}$ *can be calculated using*

$$x_j = \xi_0^{-1}(j/G).$$

Although this analysis has been performed with the fixed region (R_0, R) in mind, we can also pose it in terms of choosing a grid on an extended region. In any event, this grid formulation is, of course, of no value unless the true solution is known a priori. For our situation, a great deal of information is known about the asymptotic decay rate of u, the solution to the unbounded problem. Hence, we may use this information to generate the grid. As the domain is extended, the true solution u_R to the BBVP asymptotically converges to u. Hence the farther out we extend, the more "optimal" the grid extension should become if we use the asymptotic behavior of u in (39). For this reason we will call this grid generation/extension transformation spherically symmetric asymptotically optimal (SPAO). The following theorem summarizes the SPAO concept.

THEOREM 7. *Using the asymptotic far field form of the true solution to the* UBVP (*under the hypothesis of Theorem* 1), *the* SPAO *grid transformation to a uniform grid on* $[0, 1]$ *is given by*

$$\xi_0(r) = \frac{R_0^{-\eta} - r^{-\eta}}{R_0^{-\eta} - R^{-\eta}}, \qquad 0 < \eta < 1/3. \tag{40}$$

Proof. From the previous section and the assumptions on the UBVP, we have that $a(r)\to 1$ and $u(r)\to r^{-\gamma}$ $(\frac{1}{2}<\gamma<1)$ as $r\to\infty$.

Hence

$$\int_{R_0}^{r}\left[\frac{\rho^2(t)}{a(t)}t^2\right]^{1/3}dt\sim\int_{R_0}^{r}[t^{-2(2+\gamma)}t^2]^{1/3}\,dt$$
$$=C(R_0^{-(2\gamma-1)/3}-r^{-(2\gamma-1)/3}).$$

By (39), this gives

$$\xi_0(r)=\left(\frac{R_0^{-\eta}-r^{-\eta}}{R_0^{-\eta}-R^{-\eta}}\right)$$

asymptotically as R_0, $R\to\infty$, where $\eta=(2\gamma-1)/3<1/3$. $\square$

The restriction $\gamma>1/2$ is imposed so that $\eta>0$, which is necessary for the grid spacing to increase with r. The grid may be computed directly using

$$(41)\qquad r(z)=R_0(1-\delta z)^{-1/\eta},\qquad \delta=1-\left(\frac{R_0}{R}\right)^{\eta}.$$

That is, $x_j=r(j/G)$, for $j=0,1,2,\cdots,G$. See Fig. 1 for an example of the SPAO transformation.

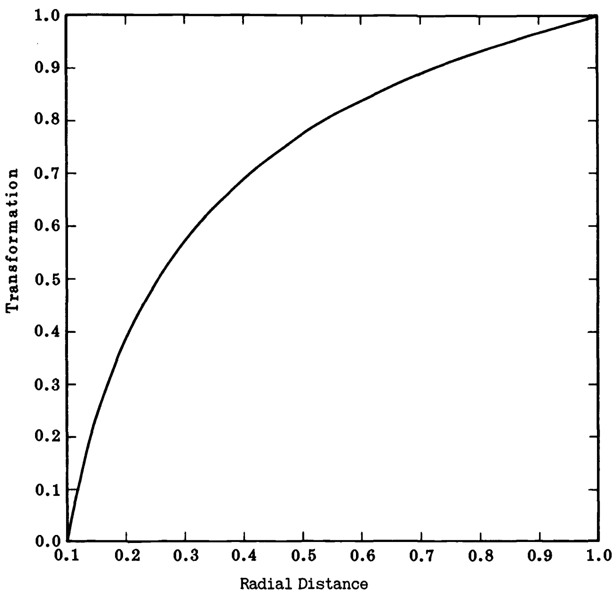


FIG. 1. *A sample* SPAO *grid transformation*; $\eta=1/3$ *and the truncated domain is* $[1,10^{-1}]$.

The above approach is a plausible grid generation strategy, and will generate a grid with better approximation properties than the standard quasi-uniform one presented in § 3.2. Before a proof is given, two lemmas are needed.

Any nonuniform grid transformation tailored to the solution of UBVPs must have the element size h_e grow as we move farther out in the mesh. We take advantage of the fact that u (and hence its derivatives) decays fairly rapidly as R increases. The next lemma gives explicit rates of decay of the derivatives of u.

LEMMA 2. *Let u satisfy the* UBVP *with $s > n/p + t + 1$. Then if Ω_e is the spherical shell generated by (x_{e-1}, x_e) we have*

$$|u|^2_{t,\Omega_e} \leqq C\|u\|_{2,s,\delta} x_e^{3-2r} \qquad \forall \tau < t+1 \tag{42}$$

where C is independent of u and Ω_e.

Proof. From the definition of $M^p_{s,\delta}$

$$D^\alpha u \in M^p_{s-t,\delta+t} \quad \text{when } |\alpha| = t,$$

and the associated decay rate properties [7]

$$\|(D^\alpha u)\sigma^\tau\|_\infty \leqq C\|D^\alpha u\|_{p,s-t,\delta+t} < C\|u\|_{p,s,\delta}$$

for all $\tau < n/p + \delta + t = n/p + [n/q - 2 - (n-\beta-2)/2] + t < t+1$. Hence,

$$|D^\alpha u(x)| \leqq C\|u\|_{2,s,\delta}\sigma^{-\tau}(x) \quad \forall \tau < t+1.$$

Integrating, we get

$$\begin{aligned} |u|^2_{t,\Omega_e} &\leqq C\|u\|^2_{2,s,\delta} \int_{x_{e-1}}^{x_e} \sigma^{-2\tau}(r) r^2\, dr \\ &\leqq C\|u\|^2_{2,s,\delta} \int_{x_{e-1}}^{x_e} r^{-2r+2}\, dr \\ &\leqq C\|u\|^2_{2,s,\delta} x_e^{3-2r} \quad \forall \tau < t+1. \end{aligned}$$ □

The next lemma describes a property of the grid transformation kernel that is then used to determine the approximation properties of the SPAO finite-element grid.

LEMMA 3. *Let $\gamma > 0$, $0 < \delta = 1 - (R_0/R)^\eta < 1$ and $\eta > 0$ hold. Then*

$$\sum_{e=1}^{N} \left(1 - \delta\frac{e}{N}\right)^\gamma \leqq \frac{2}{\delta(1+\gamma)}. \tag{43}$$

Proof. Since $\gamma > 0$, $f(z) = (1-\delta z)^\gamma$ is a decreasing function on [0, 1] (see Fig. 2), the sum in (43) can be considered a lower Riemann sum,

$$\begin{aligned} \sum_{e=1}^{N} \left(1 - \delta\frac{e}{N}\right)^{-\gamma} &\leqq \int_0^1 (1-\delta z)^\gamma\, dz \\ &= \frac{1}{\delta(\gamma+1)}[(1-\delta)^{\gamma+1} - 1] \\ &\leqq \frac{2}{\delta(\gamma+1)}. \end{aligned}$$ □

We can now compute the approximation properties of the SPAO finite-element grid.

THEOREM 8. *Let u be the solution to the* UBVP *under the assumptions of Theorem 1 with $s > 7/2 + k$, where k is the element degree. Then*

$$\inf_{\chi \in S^h} \|u - \chi\|_{SE,(R_0,R)} \leqq CN^{-k}\|u\|_{2,s,\delta} \tag{44}$$

where C is independent of N, R, and u.

Proof. By using techniques similar to those in Seager and Carey [17, Lem. 1], it is easy to show that if $\chi \in S^h$

$$\|u - \chi\|^2_{SE,(R_0,0)} \leqq C \sum_{e=1}^{G} h_e^{2k}|u|^2_{k+1,\Omega_e}. \tag{45}$$

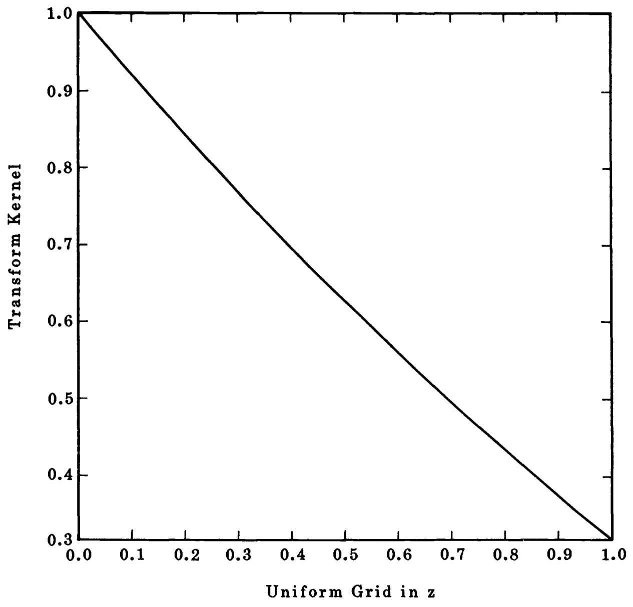


FIG. 2. *Kernel of the grid transformation.*

Next we derive an accurate bound on h_e. Set

$$h_e = x_e - x_{e-1} = R_0\left[\frac{1}{(1-\delta e/N)^{1/\eta}} - \frac{1}{(1-\delta(e-1)/N)^{1/\eta}}\right].$$

Using the bound $|b-a| \leqq 1/\alpha|b^\alpha - a^\alpha|$ for $a, b, \alpha > 1$,

$$\begin{aligned} h_e &\leqq \eta R_0\left[\frac{1}{(1-\delta e/N)} - \frac{1}{(1-\delta(e-1)/N)}\right] \\ &\leqq \frac{\eta R_0}{N}\left(1-\delta\frac{e}{N}\right)^{-2} \\ &\leqq \frac{\eta R_0}{N} x_e^{2\eta}. \end{aligned}$$

This bound has the type of behavior we would expect. It decreases linearly with N and increases (in a way similar to the mesh itself) as we move farther out into the mesh.

Applying Lemma 2 with $t = k+1$ we get

$$|u|^2_{k+1,\Omega_e} \leqq C\|u\|^2_{2,s,\delta} x_e^{3-2r} \quad \forall \tau < k+2.$$

Hence, we may write (45) as

$$\|u-\chi\|^2_{SE,(R_0,R)} \leqq C\|u\|^2_{2,s,\delta} N^{-2k} \sum_{e=1}^{G} x_e^{3-2r+4\eta k}. \tag{46}$$

The sign of the exponent on x in this expression is significant. In order to determine it, let $\tau \cong k+2$ and $\eta \cong 1/3$. Then

$$\begin{aligned} \alpha &= 3-2r+4\eta k \simeq 3-2k-4+4k/3 \\ &= -1-2k/3 \end{aligned}$$

is negative for elements (i.e., $k \geqq 1$). Thus, invoking Lemma 3 with $\gamma = -\alpha/\eta$ we bound (46) by

$$\|u - \chi\|^2_{SE,(R_0,R)} \leqq C\|u\|^2_{2,s,\delta} N^{-2k}. \qquad \square$$

This theorem extends the optimal finite-element approximation properties in terms of the parameter of interest $1/N$ to the spherically symmetric norms of the SPAO grid. With this result in hand, we can combine the error estimates due to finite-element approximation and to domain truncation, writing

$$\|u - u_h\|_{SE,(R_0,R)} = O(R^{-\gamma}) + O(h^k)$$

and then $O(h^k) \sim O(N^{-k})$. Again, this is not a completely satisfying result because there are two asymptotic parameters in the estimate. If we require the finite-element discretization error to be of the same "order" as the boundary truncation error, then $R^{-\gamma} = N^{-k}$ and hence, $N = R^{\gamma/k}$. It is clear from the above analysis that any grid generation technique that yields the optimal convergence properties of the finite-element approximation will lead to a relationship between N and R of this form. This result differs from that derived in Seager and Carey [17] in that we are now working with N instead of $h = \max h_e$.

Remark. For $\gamma = 3/2$ this is the same result as in Goldstein [11] except that the grid structures in the two approaches are quite different.

A summary of the main results is given in the following theorem.

THEOREM 9. *Let the conditions of Theorem* 1 *hold with $s > 7/2 + k$ and the right-hand endpoint R be given. If a* SPAO *grid of $N = R^{\gamma/k}$ degrees of freedom is constructed via the formula*

$$x_e = R_0\left[1 - \delta\frac{e}{G}\right]^{-1/\eta}, \qquad e = 1, \cdots, G$$

where $\delta = 1 - (R_0/R)^\eta$, and $\eta < 1/3$. Then the total spherical energy error satisfies

$$(47) \qquad \|u - u_h\|_{SE,(R_0,R)} \leqq CN^{-k}\|u\|_{2,s,\delta}. \qquad \square$$

4. Sample numerical results. As a test problem we considered problem (1) with $A = 1$, $c = B^2(B^2 + r^2)^{-2}$, and $f = 4DB^3(B^2 + r^2)^{-5/2}$. The exact solution on $0 < r < \infty$ is $u(r) = D(1 + (r/B)^2)^{-1/2}$. Both the Dirichlet condition $u(R) = 0$ and the Robin condition

$$u(R) + \frac{1}{R}\frac{\partial u}{\partial r}(R) = 0$$

on the truncated domain $0 < r < R$ are applied. The asymptotic decay rates with R for the Dirichlet and Robin conditions are $O(R^{-1})$ and $O(R^{-3})$, respectively. Thus the solution to this test problem is smooth as R is increased.

In the numerical experiment the problem was solved approximately using finite elements on the sequence of truncated domains $(0, R)$ with $R = 1$, 5, 10, 20, and 30. The spherically asymptotically optimal grids were refined for each new R until the error norms stabilized (which implies that the finite element discretization error is then negligible). Table 1 and Fig. 3 display the results for this problem with Dirichlet boundary condition and norms taken on $(0, R)$. These can be compared with the results in Table 2 and Fig. 4 where the norms are computed on the fixed subregion $(0, 1)$.

Next the numerical experiments were repeated using the Robin boundary condition and similar results summarized in Tables 3 and 4 and Figs. 5 and 6. The RATE in the tables is the computed rate of convergence with R, determined by the calculated error

TABLE 1
Error norms on $(0, R)$ *with* $U(R) = 0$.

R	N	H^1	L^∞
1	4	3.98E−1	6.25E−1
5	60	1.22E0	1.94E−1
10	146	1.77E0	9.93E−2
20	472	2.54E0	4.99E−2
30	606	3.13E0	3.33E−2
RATE (R)		+0.61	−0.86
FIT		(0.9970)	(0.9966)

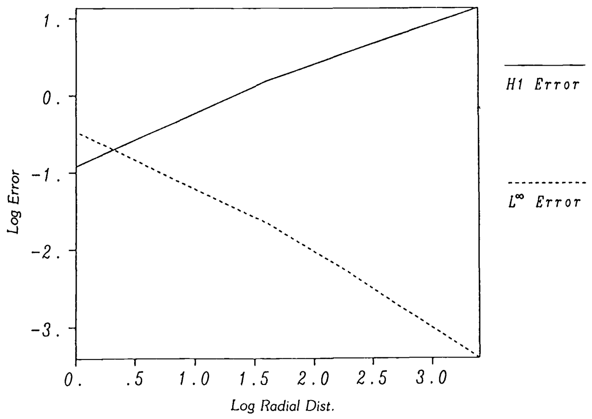


FIG. 3. *Graph of error behavior in Table* 1.

TABLE 2
Error norms on $(0, 1)$ *with* $U(R) = 0$.

R	N	H^1	L^∞
1	4	3.98E−1	6.25E−1
5	60	7.11E−2	1.51E−1
10	146	4.25E−2	7.30E−2
20	472	2.01E−2	3.60E−2
30	606	1.33E−2	2.36E−2
RATE (R)		−0.99	−0.97
FIT		(0.9989)	(0.9991)

norms and the FIT quantity in parenthesis below the RATE is the goodness-of-fit statistic. Here FIT$\sim$1 so the data is a consistent representation of the asymptotic behavior in R. For the Dirichlet case $u = O(R^{-1})$ and, as might be anticipated, Fig. 3 indicates failure to converge in $H^1(O, R)$ as R increases. The rates on the fixed subregions are much better as seen in Fig. 4. The Robin boundary condition produces exceptionally good rates of convergence with respect to R since there is a fortuitous cancellation in the analytic boundary condition for this problem using Robin data—the boundary condition is $O(R^{-3})$ rather than $O(R^{-2})$ as noted previously. This is, of course, a special case and tests with rougher problems still produce good results but

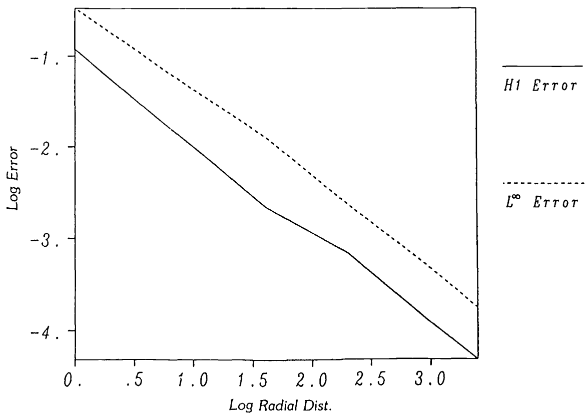


FIG. 4. *Graph of error behavior in Table* 2.

TABLE 3
Error norms on $(0, R)$ *with Robin boundary condition at* R.

R	N	H^1	L^∞
1	4	1.76E−1	2.68E−1
5	60	4.25E−2	6.80E−3
10	146	1.65E−2	9.03E−4
20	472	6.15E−3	1.21E−4
30	606	3.40E−3	3.60E−5
RATE (R)		−1.16	−2.70
FIT		(0.9921)	(0.9657)

TABLE 4
Error norms on $(0, 1)$ *with Robin boundary condition at* R.

R	N	H^1	L^∞
1	4	1.76E−1	2.68E−1
5	60	3.58E−3	5.36E−3
10	146	4.42E−4	6.93E−4
20	472	5.04E−5	8.75E−5
30	606	4.98E−5	2.73E−5
RATE (R)		−1.89	−2.71
FIT		(0.9922)	(0.9986)

not as accurate as in the present instance. Other examples and details are given in Seager [16].

5. Concluding remarks. Spherically symmetric problems on unbounded domains arise in certain areas of mathematical physics and are frequently treated by domain truncation. That is, an approximate problem is solved on a large but finite subdomain. In the present study, we have developed an analysis of the finite-element discretization and domain-truncation asymptotic errors for this class of spherical problems. A domain extension strategy is introduced to balance domain error and mesh error. This leads to the idea of spherically asymptotic optimal grids. Properties of the grid transformation,

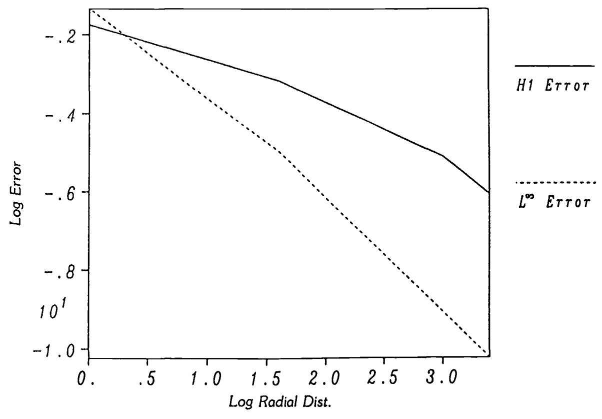


FIG. 5. *Graph of error behavior in Table* 3.

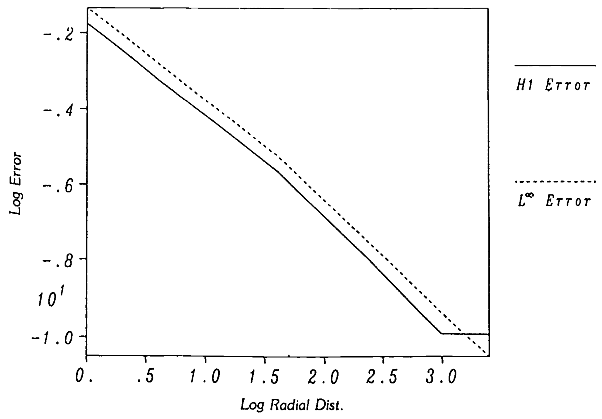


FIG. 6. *Graph of error behavior in Table* 4.

error indicators for adaptive grid refinement, and related asymptotic estimates are described. The analysis presented here provides a theoretical framework for domain extension and gridding that can be utilized in future analysis studies and applied to guide domain selection and discretization.

REFERENCES

[1] I. BABUŠKA, *The finite element method for infinite domains*, I, Math. Comp., 26 (1972), 117, pp. 1-11.

[2] I. BABUŠKA AND W. RHEINBOLDT, *Analysis of optimal finite element meshes in R^1*, Math. Comp., 33 (1979), 146, pp. 435-463.

[3] A. BAYLISS, M. GUNZBERGER, AND E. TURKEL, *Boundary conditions for the numerical treatment of elliptic equations in exterior domains*, SIAM J. Appl. Math., 42 (1982), pp. 432-451.

[4] A. BAYLISS, C. GOLDSTEIN, AND E. TURKEL, *An iterative method for the Helmholtz equation*, ICASE Tech. Report 82-4, 1982.

[5] M. CANTOR, *Spaces of functions with asymptotic conditions*, Indiana Univ. Math. J., 24 (1974), pp. 897-902.

[6] ———, *Elliptic operators and the decomposition of tensor fields*, Bull. Amer. Math. Soc., 5 (1981), pp. 235-262.

[7] ———, *Numerical treatment of potential type equations on R^3: Theoretical considerations*, SIAM J. Numer. Anal., 20 (1983), pp. 72-85.

[8] B. ENGQUIST AND A. MAJDA, *Radiative boundary conditions for acoustic and elastic wave calculations*, Comm. Pure Appl. Math., 32 (1979), pp. 312-358.

[9] G. FIX AND S. MARIN, *Variational methods for underwater acoustic problems*, J. Comput. Phys., 28 (1978), pp. 253-270.

[10] C. GOLDSTEIN, *The finite element method with non-uniform mesh sizes applied to the exterior Helmholtz problem*, Numer. Math., 38 (1981), pp. 61-82.

[11] ———, *The finite element method with non-uniform mesh sizes for unbounded domains*, Math. Comp., 36 (1981), pp. 387-404.

[12] J. KAZDEN AND F. WALKER, *Existence and conformal deformation of metrics with prescribed Gaussian and scalar curvatures*, Ann. of Math., 101 (1975), pp. 317-331.

[13] P. LAX AND R. PHILLIPS, *Scattering theory*, Rocky Mountain J. of Math., 1 (1971), pp. 173-223.

[14] L. NIRENBERG AND H. WALKER, *The null spaces of partial differential operators in R^n*, J. Math. Anal. Appl., 42 (1973), pp. 271-301.

[15] R. SCHREIBER AND S. EISENSTAT, FEM *for spherically symmetric elliptic equations*, SIAM J. Numer. Anal., 18 (1981), pp. 546-558.

[16] M. K. SEAGER, *Adaptive finite element grid generation and extension for elliptic problems posed on unbounded domains*, Ph.D. Dissertation, Univ. of Texas, May 1984; also Lawrence Livermore National Laboratory Report UCRL-53519.

[17] M. K. SEAGER AND G. F. CAREY, *Adaptive refinement for spherically symmetric problems*, University of Texas, Center for Numerical Analysis, Tech. Report 1989 (in press).

[18] L. SMARR, *Space times generated by computer black holes with gravitational radiation*, Ann. New York Acad. Sci., 302 (1976), pp. 564-604.

[19] H. WALKER, *On the null-spaces of elliptic partial differential equations in R^n*, Trans. Amer. Math. Soc., 173 (1972), pp. 263-275.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 1, pp. 112-122, January 1990

ANALYTIC CONTINUATION BY THE FAST FOURIER TRANSFORM*

JOEL FRANKLIN†

Abstract. The ill-posed problem of analytic continuation is regularized by a prescribed bound. A simple computer algorithm is given that is based on the fast Fourier transform. The algorithm computes m complex values and a positive error bound with time complexity $O(m \log m)$. As a function of the data errors and the prescribed bound, the numerical error is shown to be consistent with that prescribed by the three-circles principle of Hadamard.

Key words. analytic continuation, fast Fourier transform, ill posed

AMS(MOS) subject classifications. 30B40, 65E05, 65M30

1. Introduction.

Analytic continuation is an ill-posed problem because the solution depends discontinuously on the data.

Example 1. Let $f(z)$ be analytic for $1 \leqq |z| \leqq R$. Given that $|f(z)-z| \leqq \varepsilon$ for $|z|=1$, the problem is to compute $f(z)$ in the rest of the annulus:

If $N^{-1} < \varepsilon$, two possible solutions are $f(z) = z \pm N^{-1} z^N$. If $\varepsilon > 0$ and N is very large, the two solutions may differ greatly. Miller [11] has observed that analytic continuation can be regularized by using the three-circles theorem of Hadamard:

THEOREM. *Let $\phi(z)$ be analytic for $1 < |z| < R$ and continuous for $1 \leqq |z| \leqq R$. Let $\mu(\rho) = \max |\phi(z)|$ for $|z| = \rho$. Then $\log \mu(\rho)$ is a convex function of $\log \rho$. Thus, if $1 < r < R$ and if $\theta = (\log r)/\log R$, then*

$$\mu(r) \leqq \mu(1)^{1-\theta} \mu(R)^{\theta}. \tag{1.1}$$

Hardy proved an analogous theorem for an L^2 norm instead of the maximum norm $\mu(\rho)$.

Example 2. As before, let $f(z)$ be analytic for $1 \leqq |z| \leqq R$, and let $|f(z)-z| \leqq \varepsilon$ for $|z|=1$. Now assume $|f(z)| \leqq \beta$ for $|z|=R$. The problem is again to compute $f(z)$ for $1 < |z| < R$.

If $f_1(z)$ and $f_2(z)$ are two possible solutions and $\phi(z) = f_1(z) - f_2(z)$, then Hadamard's theorem implies, for $\theta = (\log r)/\log R$,

$$|\phi(z)| \leqq 2\varepsilon^{1-\theta} \beta^{\theta}. \tag{1.2}$$

Therefore, unless z is near the outer boundary, the difference between two possible solutions must be small.

The annulus is doubly connected. Consider, now, a simply connected region, D. Suppose $f(z)$ is analytic in D. Let $g(z)$ be given data such that

$$|g(z) - f(z)| \leqq \varepsilon \quad \text{for} \quad z \in S, \tag{1.3}$$

where S is an arc in D. The problem is to compute $f(z)$ as accurately as possible in $D - S$. Note that $D - S$ is doubly connected. By conformal mapping we can reduce this problem to the problem for the annulus and apply the three-circles theorem.

Example 3. Let $R > 1$. Let D be the elliptical domain

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} < 1, \tag{1.4}$$

* Received by the editors November 2, 1987; accepted for publication (in revised form) January 24, 1989.

† Applied Mathematics 217-50, Firestone Laboratory, California Institute of Technology, Pasadena, California 91125.

where

$$(1.5) \qquad a=\tfrac{1}{2}(R+R^{-1}), \qquad b=\tfrac{1}{2}(R-R^{-1}).$$

Let S be the interior arc $-1 \leqq x \leqq 1$, and $y=0$. For z in S let $g(z)$ be given data. Let the unknown function $f(z)$ be analytic in $\bar{D}$ and assume $|g(z)-f(z)| \leqq \varepsilon$ for z in S. The problem is to compute $f(z)$ in $D-S$.

In its present form the problem is again ill posed and a bound $|f(z)| \leqq \beta$ for z on D is required to obtain a computational solution. If we use the conformal mapping $z=\frac{1}{2}(w+w^{-1})$, then $D-S$ corresponds to the annulus $1<|w|<R$ and the segment S corresponds to the unit circle $|w|=1$. If we set $F(w)=f(z)$, $G(w)=g(z)$, then we have

$$(1.6) \qquad |G(w)-F(w)| \leqq \varepsilon \quad \text{for } |w|=1,$$

and

$$(1.7) \qquad |F(w)| \leqq \beta \quad \text{for } |w|=R.$$

Now the solution $F(w)$ can be determined as in Example 2, and every two possible solutions satisfy

$$(1.8) \qquad |F_1(w)-F_2(w)| \leqq 2\varepsilon^{1-\theta}\beta^{\theta},$$

where $\theta=(\log|w|)/\log R$.

Miller's method for solving the regularized analytic-continuation problem depends on a general principle of least squares for ill-posed problems. His computer algorithm, SNAC, uses a finite-dimensional least-squares computation. Thus, to produce m solution values, his algorithm requires $O(m^3)$ arithmetic operations.

The present paper uses a different principle, which was motivated by some earlier work [5]. The computer algorithm uses the fast Fourier transform; see [4] and [8]. Thus, to produce m solution values, the algorithm requires $O(m \log m)$ arithmetic operations.

As Example 3 illustrates, the present method may depend on a preliminary conformal mapping of a doubly connected region into an annulus. Wegmann [16] has recently published an efficient computational method for the conformal mapping of doubly connected regions that can be used to implement the present method for analytic continuation. The original region D may be replaced by an approximate subregion D' for the purpose of regularization. If $f(z)$ is analytic in D, and if $|f(z)| \leqq \beta$ in D, then $f(z)$ is analytic and has the same bound in every subregion D'. One may choose D' to include the given arc S and to include as much of the rest of D as is conveniently possible.

Other approaches to numerical analytic continuation have been made by Bisshop [1], Niethammer [12], Stefanescu [14], and Reichel [13]. One may also state the problem as a Fredholm integral equation of the first kind, to which one may apply Tikhonov's method (see Tikhonov and Arsenin [15]). For a discussion of the error in Tikhonov's method see the section in [6] on harmonic continuation, which is equivalent to analytic continuation.

Analytic continuation from an arc is equivalent to the Cauchy problem for Laplace's equation, for which there exists a vast amount of literature. General references include the books by Hadamard [7], Tikhonov and Arsenin [15], Lavrentiev [10], and Carasso and Stone [3]. Logarithmic convexity and ill-posed problems are discussed by Knops [9].

Whereas the three-circles theorem proves logarithmic convexity for the maximum norm, Miller [11] used logarithmic convexity for a quadratic norm on the m-dimensional complex linear space; we shall do likewise. In the limit as $m \to \infty$, this norm becomes the continuous L^2 norm.

2. The problem for an annulus. We assume that an unknown function $f(z)$ has a Laurent series

$$f(z) = \sum_{k=-\infty}^{\infty} c_k z^k \qquad (1 \leqq |z| \leqq R) \tag{2.1}$$

that is absolutely convergent on the bounding circles $|z| = 1$ and $|z| = R$. We are given numerical values g_j approximating $f(z)$ on the unit circle. Let m be a power of 2, and let $\omega = \exp(2\pi i/m)$. We assume

$$\frac{1}{m} \sum_{j=0}^{m-1} |g_j - f(\omega^j)|^2 \leqq \varepsilon^2, \tag{2.2}$$

where ε is a known positive bound for the data error. We are also given a positive bound β for a quadratic norm of f on the outer boundary:

$$\frac{1}{m} \sum_{j=0}^{m-1} |f(R\omega^j)|^2 \leqq \beta^2. \tag{2.3}$$

Finally, we are given a positive bound τ_m for the truncation error of the Laurent series. We assume

$$\sum_{k<-m/2} |c_k| R^{m/2-1} + \sum_{k \geqq m/2} |c_k| R^k \leqq \tau_m. \tag{2.4}$$

In summary, we are given the following: the integer m, where m is a power of 2 greater than 1; the complex numbers $g_0, g_1, \cdots, g_{m-1}$; the positive numbers ε, β, and τ_m; two radii, r and R, where $1 < r < R$. As a rule, the numbers ε and τ_m will be small; the number β will be moderate or large.

The problem is to compute the unknown $f(z)$ on the interior circle $|z| = r$. For $|z| = r$ we will compute complex numbers $b_0, \cdots, b_{m-1}$ approximating the unknowns $f(r\omega^j)$ $(j = 0, \cdots, m-1)$.

In the analysis of the algorithm, we will prove an inequality for the error norm μ defined by

$$\mu^2 = \frac{1}{m} \sum_{j=0}^{m-1} |b_j - f(r\omega^j)|^2. \tag{2.5}$$

We will show that the error norm μ satisfies

$$\mu \leqq \tau_m + 2\varepsilon^{1-\theta}(\beta + \varepsilon + \tau_m)^\theta, \tag{2.6}$$

where $\theta = (\log r)/\log R$. (Actually, we shall get a somewhat better result.) Thus, as a function of the data-error bound, ε, the solution-error bound, μ, is of the order $\varepsilon^{1-\theta}$. For example, if r is near 1, then μ behaves about like ε; if r is near $\sqrt{R}$, μ behaves like $\sqrt{\varepsilon}$; but if r is near R, we obtain $\mu \leqq \tau_m + 2(\beta + \varepsilon + \tau_m)$, which is of academic interest only.

For fixed r, the algorithm produces the positive error bound μ_1 and m complex numbers $b_0, \cdots, b_{m-1}$. Using the fast Fourier transform, the algorithm has time complexity of the order of $m \log m$.

3. The algorithm. As described in the last section, we are given the data

$$m;\ g_0, \cdots, g_{m-1};\ \varepsilon, \beta, \tau_m;\ r, R. \tag{3.1}$$

The algorithm will compute m complex numbers, $b_0, \cdots, b_{m-1}$, and a positive number, μ_1.

Notation. Let $\mathbf{u}$ be any vector with m complex components. By the equation

$$\mathbf{v} = F\mathbf{u} \tag{3.2}$$

we shall mean that $\mathbf{v}$ is the finite Fourier transform of $\mathbf{u}$:

$$v_j = \sum_{k=0}^{m-1} u_k \omega^{jk} \qquad (j = 0, \cdots, m-1) \tag{3.3}$$

where $\omega = \exp(2\pi i/m)$. Equivalently, we may write $\mathbf{u} = F^{-1}\mathbf{v}$, the inverse transform of $\mathbf{v}$:

$$u_k = \frac{1}{m} \sum_{j=0}^{m-1} v_j \omega^{-kj} \qquad (k = 0, \cdots, m-1). \tag{3.4}$$

The algorithm uses the data (3.1) and the auxiliary variables β_1, λ, θ, and $\mathbf{u}$ to compute the vector $\mathbf{b}$ and the positive number μ_1. (The components b_j approximate $f(r\omega^j)$; the number μ_1 is an upper bound for the error norm μ.)

```
Algorithm Analytic Continuation;
Begin
    $\theta := (\log r)/\log R$;
    $\beta_1 := \beta + \varepsilon + \tau_m$;
    $\lambda := \frac{\varepsilon}{\beta_1}\frac{\theta}{1-\theta}$;
    $\mathbf{u} := F^{-1}\mathbf{g}$;                    {inverse fft}
    for $k := 0$ to $\frac{m}{2} - 1$ do
        $u_k := \frac{r^k}{1 + \lambda R^k} u_k$;
    for $k := \frac{m}{2}$ to $m - 1$ do
        $u_k := r^{k-m} u_k$;
    $\mathbf{b} := F\mathbf{u}$;                         {fft}
    $\mu_1 := \tau_m + (\varepsilon + \lambda\beta_1)\lambda^{-\theta}$
end.
```

4. Analysis of the algorithm. Let $f(z)$ be an analytic function in the annulus $1 < |z| < R$ and r be a *fixed* radius satisfying $1 < r < R$. Given the integer m; positive numbers r, R, ε, β, τ_m; and complex numbers $g_0, \cdots, g_{m-1}$, the algorithm computes complex numbers $b_0, \cdots, b_{m-1}$ to approximate the unknown values $f(r\omega^j)$ $(j = 0, \cdots, m-1)$, where $\omega = \exp(2\pi i/m)$. We assume m is a power of 2 greater than 1.

Time complexity. The algorithm uses the fast Fourier transform twice and $O(m)$ other operations. Therefore the algorithm has time complexity $T(m) = O(m \log m)$.

Error analysis. We now analyze the numerical error, $b_j - f(r\omega^j)$ $(j=0,\cdots,m-1)$. We will show that the error is bounded in terms of the given numbers ε, β, and τ_m, which are defined in the text that follows.

If $\mathbf{v}$ is a vector with complex components $v_0,\cdots,v_{m-1}$, then the L_2 norm is

$$\|\mathbf{v}\| = \left(m^{-1}\sum_{j=0}^{m-1}|v_j|^2\right)^{1/2}. \tag{4.1}$$

Let x vary in the interval $1 \leqq x \leqq R$, and define the vector $\mathbf{f}(x)$ with m complex components $f(x\omega^j)$ $(j=0,1,\cdots,m-1)$. Assume

$$\|\mathbf{f}(1)-\mathbf{g}\| \leqq \varepsilon, \tag{4.2}$$

where ε is a given positive bound for the data error.

On the outer circle, $|z|=R$, we assume the bound

$$\|\mathbf{f}(R)\| \leqq \beta, \tag{4.3}$$

which regularizes the ill-posed problem of analytic continuation. For a positive data error ε, the values of the unknown vector $\mathbf{f}(r)$ *must* depend on the outer bound β.

We assume that the unknown function $f(z)$ has an absolutely convergent Laurent series (2.1) and that the truncation error has the bound

$$\sum_{k<-m/2}|c_k|R^{m/2-1} + \sum_{k\geqq m/2}|c_k|R^k \leqq \tau_m. \tag{4.4}$$

This implies

$$\left|f(z) - \sum_{-n\leqq k<n} c_k z^k\right| \leqq \tau_m \tag{4.5}$$

in the closed annulus $1 \leqq |z| \leqq R$, where $n = m/2$.

THEOREM. *Under the preceding assumptions, define*

$$\theta = (\log r)/\log R, \qquad \beta_1 = \beta+\varepsilon+\tau_m \quad \textit{and} \quad \lambda = \frac{\varepsilon}{\beta_1}\frac{\theta}{1-\theta}, \tag{4.6}$$

which implies that $0<\theta<1$, $\beta_1>0$, $\lambda>0$. *For* $k=-n,\cdots,n-1$ *define*

$$G_k = m^{-1}\sum_{j=0}^{m-1} g_j\omega^{-kj}. \tag{4.7}$$

For $j=0,\cdots,m-1$ *define*

$$b_j = \sum_{-n\leqq k<0} G_k r^k\omega^{jk} + \sum_{0\leqq k<n} G_k r^k(1+\lambda R^k)^{-1}\omega^{jk}. \tag{4.8}$$

Also define the constant $1<C\leqq 2$ *by*

$$C = (1-\theta)^{-(1-\theta)}\theta^{-\theta}. \tag{4.9}$$

Then the numerical error satisfies

$$\|\mathbf{b}-\mathbf{g}(r)\| \leqq \tau_m + C\varepsilon^{1-\theta}\beta_1^{\theta}. \tag{4.10}$$

This completes the theorem whose proof will require the following elementary result.

LEMMA. *Assume* $x>0$, $p_i>0$, q_i *real. Then*

$$\log\sum_{k=1}^{N} p_k x^{q_k} \textit{ is a convex function of } \log x.$$

Proof. Set $x=e^t$ and call the sum $S(t)$. To prove $\log S(t)$ convex, it suffices to prove

$$S(t)^2 \leqq S(t-h)S(t+h) \tag{4.11}$$

for all real t and h. We have

$$\begin{aligned} S(t) &= \sum p_k \exp(q_k t) \\ &= \sum p_k[\exp \tfrac{1}{2}q_k(t-h)] \cdot [\exp \tfrac{1}{2}q_k(t+h)]. \end{aligned}$$

Since $p_k > 0$, the inequality (4.11) follows from the Schwarz inequality. □

Proof of the Theorem. Define the m complex numbers

$$A_k = m^{-1} \sum_{j=0}^{m-1} f_j(1)\omega^{-kj} \qquad (-n \leqq k < n), \tag{4.12}$$

where, as usual, $n = m/2$. Similarly, define the numbers

$$G_k = m^{-1} \sum_{j=0}^{m-1} g_j \omega^{-kj} \qquad (-n \leqq k < n). \tag{4.13}$$

For $1 \leqq x \leqq R$ and $j = 0, \cdots, m-1$, define the following functions of x:

$$a_j(x) = \sum_{-n \leqq k < n} A_k x^k \omega^{jk}, \tag{4.14}$$

$$\phi_j(x) = \sum_{-n \leqq k < 0} A_k x^k \omega^{jk} + \sum_{0 \leqq k < n} A_k x^k (1 + \lambda R^k)^{-1} \omega^{jk}, \tag{4.15}$$

$$b_j(x) = \sum_{-n \leqq k < 0} G_k x^k \omega^{jk} + \sum_{0 \leqq k < n} G_k x^k (1 + \lambda R^k)^{-1} \omega^{jk}. \tag{4.16}$$

For the given $x = r$, we get the numbers $b_j(r) = b_j$ defined in (4.8). We wish to prove (4.10) for $\|\mathbf{b} - \mathbf{f}(r)\|$. First we will express the Fourier coefficients A_k in terms of the Laurent coefficients c_j. From (4.12) we have

$$f(\omega^j) = f_j(1) = \sum_{-n \leqq k < n} A_k \omega^{jk} \qquad (j = 0, \cdots, m-1). \tag{4.17}$$

But the Laurent series gives

$$f(\omega^j) = \sum_{-\infty < k < \infty} c_k \omega^{jk} \qquad (j = 0, \cdots, m-1). \tag{4.18}$$

Since $\omega^m = 1$, we may write

$$f(\omega^j) = \sum_{-n \leqq k < n} \left(\sum_{-\infty < s < \infty} c_{k+sm} \right) \omega^{jk} \qquad (j = 0, \cdots, m-1). \tag{4.19}$$

But the Fourier coefficients A_k are defined uniquely by (4.17). Therefore, (4.19) implies

$$A_k = \sum_{-\infty < s < \infty} c_{k+sm} \qquad (k = -n, \cdots, n-1). \tag{4.20}$$

Now we will determine a bound for $\|\mathbf{a}(x) - \mathbf{f}(x)\|$. From (4.14) and (4.20) we determine

$$a_j(x) = \sum_{-n \leqq k < n} \left(\sum_{-\infty < s < \infty} c_{k+sm} \right) x^k \omega^{jk} \qquad (j = 0, \cdots, m-1). \tag{4.21}$$

If we define the unique residue $k \bmod m$ in the set $-n, \cdots, n-1$, then from (4.21)

$$a_j(x) = \sum_{-\infty < k < \infty} c_k x^{(k \bmod m)} \omega^{jk} \qquad (j = 0, \cdots, m-1). \tag{4.22}$$

Subtracting the Laurent series for $f_j(x)$, we obtain

$$a_j(x) - f_j(x) = \sum_{-\infty < k < \infty} c_k [x^{(k \bmod m)} - x^k] \omega^{jk}. \tag{4.23}$$

We have $(k \bmod m) = k$ for $-n \leqq k < n$, while

$$(k \bmod m) > k \quad \text{for} \quad k < -n,$$

and

$$(k \bmod m) < k \quad \text{for} \quad k \geqq n.$$

Since $1 \leqq x \leqq R$, equation (4.23) implies that

$$|a_j(x) - f_j(x)| \leqq \sum_{k<-n} |c_k| R^{n-1} + \sum_{k \geqq n} |c_k| R^k \leqq \tau_m \tag{4.24}$$

where the given bound τ_m satisfies (4.4). Therefore

$$\|\mathbf{a}(x) - \mathbf{f}(x)\| \leqq \tau_m \qquad (1 \leqq x \leqq R). \tag{4.25}$$

Next we will determine a bound for $\|\mathbf{b}(x) - \mathbf{a}(x)\|$ at $x = r$. To do so, we will determine a bound for $\|\mathbf{b}(x) - \mathbf{a}(x)\|$ at $x = 1$ and at $x = R$. We will then use the lemma, which implies that $\log \|\mathbf{b}(x) - \mathbf{a}(x)\|$ is a convex function of $\log x$, to show that

$$\|\mathbf{b}(r) - \mathbf{a}(r)\| \leqq \|\mathbf{b}(1) - \mathbf{a}(1)\|^{1-\theta} \|\mathbf{b}(R) - \mathbf{a}(R)\|^{\theta} \tag{4.26}$$

where $\theta = (\log r)/\log R$. The lemma is applicable because, by (4.14) and (4.16),

$$\|\mathbf{b}(x) - \mathbf{a}(x)\|^2 = \sum_{-n \leqq k < n} p_k x^{2k}, \tag{4.27}$$

where all p_k are positive.

First set $x = 1$. Then from (4.14) and (4.15),

$$\begin{aligned} \|\phi(1) - \mathbf{a}(1)\|^2 &= \sum_{0 \leqq k < n} |A_k \lambda R^k (1 + \lambda R^k)^{-1}|^2 \\ &\leqq \sum_{0 \leqq k < n} |A_k \lambda R^k|^2 \\ &\leqq \sum_{-n \leqq k < n} |A_k \lambda R^k|^2 = \lambda^2 \|\mathbf{a}(R)\|^2. \end{aligned} \tag{4.28}$$

But (4.25) implies, for $x = R$,

$$\|\mathbf{a}(R)\| \leqq \|\mathbf{f}(R)\| + \tau_m \leqq \beta + \tau_m, \tag{4.29}$$

where β is the given bound for $\|\mathbf{f}(R)\|$. Now (4.28) gives

$$\|\phi(1) - \mathbf{a}(1)\| \leqq \lambda(\beta + \tau_m). \tag{4.30}$$

From (4.15) and (4.16),

$$\begin{aligned} \|\mathbf{b}(1) - \phi(1)\|^2 &= \sum_{-n \leqq k < 0} |G_k - A_k|^2 + \sum_{0 \leqq k < n} |G_k - A_k|^2 (1 + \lambda R^k)^{-2} \\ &\leqq \sum_{-n \leqq k < n} |G_k - A_k|^2 = \|\mathbf{g} - \mathbf{f}(1)\|^2 \leqq \varepsilon^2. \end{aligned}$$

Thus, if ε is the given data-error bound, we have

$$\|\mathbf{b}(1) - \phi(1)\| \leqq \varepsilon. \tag{4.31}$$

Applying the triangle inequality to (4.30) and (4.31), we deduce

$$\|\mathbf{b}(1) - \mathbf{a}(1)\| \leqq \varepsilon + \lambda(\beta + \tau_m). \tag{4.32}$$

Now we will determine the bound for $\|\mathbf{b}(R)-\mathbf{a}(R)\|$. From (4.14) and (4.15) we get

$$\|\phi(R)-\mathbf{a}(R)\|^2=\sum_{0\leqq k<n}|A_kR^k\cdot\lambda R^k(1+\lambda R^k)^{-1}|^2$$
$$\leqq\sum_{-n\leqq k<n}|A_kR^k|^2=\|\mathbf{a}(R)\|^2.$$

From (4.29) we obtain

$$\|\phi(R)-\mathbf{a}(R)\|\leqq\beta+\tau_m. \tag{4.33}$$

From (4.15) and (4.16),

$$\|\mathbf{b}(R)-\phi(R)\|^2=\sum_{-n\leqq k<0}|G_k-A_k|^2R^{2k}+\sum_{0\leqq k<n}|G_k-A_k|^2R^{2k}(1+\lambda R^k)^{-2}$$
$$\leqq\sum_{-n\leqq k<0}|G_k-A_k|^2+\lambda^{-2}\sum_{0\leqq k<n}|G_k-A_k|^2.$$

Since

$$\sum_{-n\leqq k<n}|G_k-A_k|^2=\|\mathbf{g}-\mathbf{f}(1)\|^2\leqq\varepsilon^2,$$

we obtain

$$\|\mathbf{b}(R)-\phi(R)\|\leqq\varepsilon\cdot\max(1,\lambda^{-1})<\varepsilon(1+\lambda^{-1}). \tag{4.34}$$

Applying the triangle inequality to (4.33) and (4.34), we obtain

$$\|\mathbf{b}(R)-\mathbf{a}(R)\|\leqq\beta+\tau_m+\varepsilon(1+\lambda^{-1}). \tag{4.35}$$

Now we are ready to bring our results together. By (4.32) and (4.35), we have

$$\|\mathbf{b}(1)-\mathbf{a}(1)\|\leqq\varepsilon+\lambda\beta_1,\qquad\|\mathbf{b}(R)-\mathbf{a}(R)\|\leqq(\varepsilon+\lambda\beta_1)\lambda^{-1},$$

where $\beta_1=\beta+\varepsilon+\tau_m$. From the (4.26) we obtain

$$\|\mathbf{b}(r)-\mathbf{a}(r)\|\leqq(\varepsilon+\lambda\beta_1)\lambda^{-\theta}, \tag{4.36}$$

where r is the given radius satisfying $1<r<R$. Setting the variable x equal to r, we deduce from (4.25)

$$\|\mathbf{a}(r)-\mathbf{f}(r)\|\leqq\tau_m. \tag{4.37}$$

The triangle inequality yields

$$\|\mathbf{b}(r)-\mathbf{f}(r)\|\leqq\tau_m+(\varepsilon+\lambda\beta_1)\lambda^{-\theta}. \tag{4.38}$$

As a function of λ, the right-hand side is minimized by the value defined in (4.6). Then the proved (4.38) is the required inequality (4.10). This completes the proof of the theorem. □

5. Computer testing. It is easy to implement the algorithm described in § 3. Using an available fast Fourier transform (FFT) subroutine, a PASCAL program was written to test the algorithm for an example of analytic continuation from a line segment. The function

$$F(w)=\frac{1}{2-w} \tag{5.1}$$

was used and data $G(w)$ for $F(w)$ were given on the line segment $-1 \leqq w \leqq 1$. A data-error bound

$$|G(w)-F(w)| \leqq \varepsilon = 10^{-4} \tag{5.2}$$

was assumed. The data $G(w)$ are used to continue the supposedly unknown function $F(w)$ from the line segment into an ellipse with foci at ± 1.

Let E_R be the ellipse in the w-plane given by

$$\frac{u^2}{A^2}+\frac{v^2}{B^2}=1, \tag{5.3}$$

where $w=u+iv$, $A=\frac{1}{2}(R+R^{-1})$, $B=\frac{1}{2}(R-R^{-1})$. Assume $1<R<3.732$ so that the pole of $F(w)$ at $w=2$ lies outside the ellipse. A bound β is prescribed for $|F(w)|$ on the boundary E_R.

The conformal mapping $w=\frac{1}{2}(z+z^{-1})$ maps the annulus $1<|z|<R$ into the region bounded by the slit $-1 \leqq w \leqq 1$ and the ellipse E_R. For $1<r<R$ the circle $|z|=r$ is mapped into an interior confocal ellipse E_r. Values for $F(w)$ on the interior ellipse E_r are computed.

Set $F(w)=f(z)$, $G(w)=g(z)$. Thus $f(z)$ is the supposedly unknown function

$$f(z)=\frac{1}{2-\frac{1}{2}(z+z^{-1})}. \tag{5.4}$$

Let m be a large power of 2; typically, $m=256$. Let $\omega=\exp(2\pi i/m)$. The test simulates data $g(z)$ on the unit circle by computing

$$g(\omega^j)=f(\omega^j)+\varepsilon X_j \qquad (j=0,\cdots,m-1), \tag{5.5}$$

where $\varepsilon=10^{-4}$ and X_j is a computer-generated random number satisfying $-1 \leqq X_j \leqq 1$. Thus, εX_j is a simulated data error bounded by $\pm\varepsilon$.

As described in § 3, the algorithm requires an upper bound τ_m for the Laurent-series truncation error. The function $f(z)$ defined in (5.4) has the Laurent series

$$\sum_{k=-\infty}^{\infty} c_k z^k = \frac{\gamma_1}{z-z_1}+\frac{\gamma_2}{z-z_2}, \tag{5.6}$$

where z_1 and z_2 are the reciprocal poles $2 \pm \sqrt{3}$. Therefore, if $n=m/2$,

$$\sum_{k<-n} |c_k| R^{n-1} + \sum_{k \geqq n} |c_k| R^k = O((2-\sqrt{3})^n R^n) \tag{5.7}$$

and $\tau_m = O((2-\sqrt{3})^n R^n)$. If $m \geqq 256$ and $R \leqq 3$, we have $\tau_m = O(7.28 \times 10^{-13})$. Thus, within the limit of roundoff error, we may set $\tau_m=0$.

Table 1 gives the results of a numerical test. In accordance with (5.5), randomly perturbed data were given on the unit circle. The following values were fixed:

$$\varepsilon=10^{-4}, \qquad m=256, \qquad \tau_m=0, \qquad R=3, \qquad \beta=0.972. \tag{5.8}$$

TABLE 1
Precise $\beta=0.972$.

r	λ	μ_1	μ
1.25	2.62E−5	1.07E−3	5.81E−5
1.50	6.02E−5	5.72E−3	2.99E−4
1.75	1.07E−4	2.15E−2	1.14E−3
2.00	1.76E−4	6.34E−2	4.28E−3
2.25	2.90E−4	1.56E−1	1.32E−2
2.50	5.17E−4	3.32E−1	3.72E−2
2.75	1.20E−3	6.20E−1	1.02E−1

Thus, the data errors were bounded by $\pm 10^{-4}$. With the outer radius fixed at $R=3$, the norm $\|\mathbf{f}(R)\|$ is fixed at 0.972, and this value was used for β. (This upper bound may be replaced by a value twice as large without an appreciable change in the computations.)

The radius r was given the seven values 1.25, 1.50, $\cdots$, 2.75. For each r, the algorithm was applied to the randomly perturbed data $g_0, \cdots, g_{255}$ and the numbers λ, μ_1, and $b_0, \cdots, b_{255}$ were computed. For each r, using an IBM XT, the computation required approximately 4 seconds.

For each r the algorithm computed an upper bound μ_1 for the true solution error

$$\mu = \|\mathbf{b} - \mathbf{f}(r)\|. \tag{5.9}$$

In a separate computation, which used the function definition (5.4), the true error μ was computed. For the different values of r, the values of the true error μ appear in the last column of Table 1.

The effect of doubling the prescribed bound β for the norm $\|\mathbf{f}(R)\|$ on the outer circle is given in Table 2. The only surprise came for $r=2$. For that value the true error μ decreased: it went from 4.28×10^{-3} in Table 1 to 2.61×10^{-3} in Table 2. For the other tested values of r the true error increased with the use of the crude upper bound β. As a rough check of the computations, a naive analytic continuation by the FFT was performed with the parameter λ equal to zero. For inner radius $r=2$ the result was a true error $\mu = 7{\cdot}8\times 10^{32}$.

TABLE 2
Crude $\beta = 1.94$.

r	λ	μ_1	μ
1.25	1.31E−5	1.11E−3	6.72E−5
2.00	8.8E−5	6.72E−2	2.61E−3
2.75	6.0E−3	6.59E−1	1.59E−1

The algorithm was tested with other functions. $F(w) = e^w$ was continued from the interval $-1 \leqq w \leqq 1$ into the rest of the complex plane. This example has the same form as the one given above, but it is easier because $F(w)$ has no singularity in the finite plane. As before, data $G(w)$ are given for $-1 \leqq w \leqq 1$, with $|G(w) - e^w| \leqq 10^{-4}$. If the precise bound $\beta = 5.71$ is prescribed for the norm $\|\mathbf{f}(5)\|$ on the outer ellipse E_5, the algorithm computes values b_j approximating the true values of e^w on the inner ellipse E_3 with two-digit accuracy. On E_3 the computed error bound is $\mu_1 = 0.33$, but the true error μ is smaller. A typical experiment with random data errors yielded the true error $\mu = \|\mathbf{b} - \mathbf{f}(3)\| = 0.0125$.

REFERENCES

[1] F. BISSHOP, *Numerical conformal mapping and analytic continuation*, Quart. Appl. Math., 41 (1983), pp. 125–142.

[2] J. CANNON AND K. MILLER, *Some problems in numerical analytic continuation*, J. Soc. Indust. Appl. Math. Ser. B Numer. Anal., 2 (1965), pp. 87–96.

[3] A. CARASSO AND A. P. STONE, EDS., *Improperly Posed Boundary Value Problems*, Pitman, San Francisco, 1975.

[4] J. M. Cooley and J. W. Tukey, *An algorithm for the machine calculation of Fourier series*, Math. Comp., 19 (1965), pp. 297-301.

[5] J. Franklin, *Minimum principles for ill-posed problems*, SIAM J. Math. Anal., 9 (1978), pp. 638-650.

[6] ———, *On Tikhonov's method for ill-posed problems*, Math. Comp., 28 (1974), pp. 889-907.

[7] J. Hadamard, *Lectures on Cauchy's Problem in Linear Partial Differential Equations*, Dover, New York, 1952.

[8] P. Henrici, *Applied and Computational Complex Analysis*, Vol. 3, John Wiley, New York, 1986.

[9] R. J. Knops, ed., *Symposium on Non-Well-Posed Problems and Logarithmic Convexity*, Springer-Verlag, Berlin, New York, 1973.

[10] M. M. Lavrentiev, *Some Improperly Posed Problems of Mathematical Physics*, Springer-Verlag, Berlin New York, 1967.

[11] K. Miller, *Least squares methods for ill-posed problems with a prescribed bound*, SIAM J. Math. Anal., 1 (1970), pp. 52-74.

[12] W. Niethammer, *Ein numerisches Verfahren zur analytischen Fortsetzung*, Numer. Math., 21 (1973), pp. 81-92.

[13] L. Reichel, *Numerical methods for analytic continuation and mesh generation*, Constr. Approx. 2 (1986), pp. 23-39.

[14] I. S. Stefanescu, *On the stable analytic continuation with rational functions*, J. Math. Phys., 21 (1980), pp. 175-189.

[15] A. N. Tikhonov and V. Y. Arsenin, *Solutions of Ill-Posed Problems*, John Wiley, New York, 1977.

[16] R. Wegmann, *An iterative method for the conformal mapping of doubly connected regions*, J. Comput. Appl. Math., 14 (1986), pp. 79-98.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 1, pp. 123-144, January 1990

AGGREGATION METHODS FOR SOLVING SPARSE TRIANGULAR SYSTEMS ON MULTIPROCESSORS*

JOEL H. SALTZ†

Abstract. Efficient methods are presented for solving large sparse triangular systems on multiprocessors. These methods use heuristics for the aggregation, mapping, and scheduling of relatively fine-grained computations whose data dependencies are specified by directed acyclic graphs. Results of experiments run on the Encore Multimax, as well as model problem analysis, measure the performance of the partitioning strategies on shared-memory architectures with varying synchronization costs.

Key words. sparse triangular systems, shared memory, synchronization, partitioning

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

1. Introduction. Techniques are proposed for mapping solutions of very sparse triangular systems of linear equations onto a range of parallel architectures. In solutions of such systems, the number of floating point operations that can be performed at any one time is typically rather limited. Minimization of synchronization and communication overheads can consequently become particularly important. Methods for solving very sparse triangular systems are vital for efficiently parallelizing conjugate gradient type algorithms preconditioned with incomplete LU factorizations.

We focus on sparse triangular systems generated by incomplete factorizations of matrices arising from discretizations of two-dimensional partial differential equations. The techniques described here can, however, be used in any computation in which the data dependencies exhibit the appropriate underlying structure, examples include sparse incomplete numeric factorizations and sparse codes for solving dynamic programming problems.

We assign to a single processor all computations pertaining to a row of the matrix. All computations pertaining to a given row are performed as soon as possible after the data required is known to be available. Parallelism is achieved by solving for a number of rows simultaneously. The dependencies between the rows of the triangular matrix determine the amount of potential concurrency.

We preprocess the data structure representing the sparse triangular matrix in a way that identifies inter-row parallelism and allows the problem to be mapped or scheduled in various different ways. After the preprocessing, parameters are chosen that allow a variety of tradeoffs in the schedule or mapping specification. The need to amortize the cost of performing preprocessing does limit the applicability of the types of methods described here to situations, such as iterative algorithms, where we perform many computations having the same data dependencies.

In the execution of a fine-grained problem on a shared-memory machine such as the Encore Multimax, primary impediments to the achievement of ideal multiprocessor performance are (1) load imbalance, (2) synchronization delays, and (3) programming techniques that introduce computations intended to coordinate the parallel execution

* Received by the editors June 15, 1987; accepted for publication (in revised form) November 10, 1988. This work was partially supported by National Aeronautics and Space Administration contract NAS1-18107, Office of Naval Research contract N00014-86-K-0310, and National Science Foundation grant DCR 8106081.

† Department of Computer Science, Yale University, 51 Prospect Street, New Haven, Connecticut 06520-2158, and Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, Virginia 23368.

of a problem and not found in a corresponding sequential program. Our techniques provide a reduction in (1) and (2). Since a prescheduled approach is used, it will be shown that it is possible to keep (3) from becoming a serious problem. The strategies developed here for dealing with these overheads are to (a) reduce the number of synchronizations required during the solution of the problem, (b) make synchronizations less expensive, and (c) improve the load balance in between synchronizations. Due to rather slow computation and rapid synchronization, the Multimax presents a rather benign parallel environment. We will consequently make use of this machine to provide hardware simulations of algorithm performance in architectures with relatively larger synchronization times.

In message-passing environments (such as the Intel iPSC), (1) and (3) remain crucial impediments to the achievement of ideal multiprocessor performance. In current message-passing machines, communication startups are quite expensive [13]; techniques that minimize the number of such startups are consequently of crucial importance. The techniques discussed here that reduce the number of synchronizations clearly also reduce the number of startups required. In message-passing machines the amount of information communicated also can play an important role in the determination of performance. As we shall see from the analysis of a model problem below, the specification of the parameters in the parameterized mapping plays an important role in determining the amount of information that must be communicated. The mapping techniques discussed have been implemented on a message-passing machine; the experimental results are presented in [11].

The problem partitions and work schedules that result from the process described above may be viewed as a generalization of the work described by Saad and Schultz [12]. In that report, a wavefront method was proposed for scheduling work involved in forward and backsolves of matrices arising from incomplete factorizations of matrices generated by five-point discretizations of two-dimensional elliptic partial differential equations. The work described by Saad, as well as the results presented here, assume a row-oriented matrix storage scheme. Experimental work has been reported on the NYU Ultracomputer prototype involving the use of a wavefront method, where the work involved in solving for rows of sparse triangular linear systems was allocated in a self-scheduled manner [6].

A related body of literature also exists on the solution of triangular systems that are less sparse than the ones described here; these systems are generally obtained from matrix factorizations used in direct methods for solving sparse or nonsparse systems of linear equations [7], [3], [10], [4]. The very sparse triangular systems considered in this paper differ significantly from those examined in the above references. In the systems examined here, very few computations are required to solve for a given variable. Useful parallelism can be obtained because the data dependencies between rows can allow one to solve for many variables simultaneously. In [11] the performance obtained by solving for many variables simultaneously is compared with the performance obtained from performing the row substitutions sequentially and parallelizing each substitution individually.

The methods to be presented are quite similar in spirit to algorithms used to map uniform recurrence relations to systolic arrays [1]. While the techniques used by those algorithms are very different, we can obtain the same mappings using the two sets of techniques if both are presented with inputs representing two dimensional uniform recurrence relations.

In § 2 we discuss methods for generating a parameterized problem decomposition that allows considerable flexibility in determining the granularity of parallelism and facilitates inexpensive forms of synchronization. This approach defines a kind of coordinate system that can be used to specify how the problem is to be solved. In § 3 we derive expressions using the parameters from the decomposition defined in § 2 that allow us to specify a partitioning of the triangular system that guarantees that all data dependencies will be respected. The expressions describing the parametrized schedules depend on, among other things, the type of synchronization used.

In § 4, through the analysis of a model problem, we analyze the tradeoffs between load imbalance and synchronization costs, as well as the tradeoffs between load imbalance and communication costs. An inexpensive method for explicitly balancing the load is described in § 5. In § 6 we report experimental results on the Encore Multimax multiprocessor that (1) explore the effect of parametric variations on granularity on performance, (2) assess the merit of explicitly balancing load, and (3) compare the performance obtained using different synchronization techniques.

2. Problem partitioning.

2.1. Automated problem partitioning. Detailed knowledge of an algorithm's data dependencies are crucial for employing a variety of optimizations that effect the efficiency of programs on multiprocessors. In programs that use sparse data structures, the crucial data dependencies are frequently not determined until program execution.

The *C* program segment 1 in Fig. 1 solves a lower triangular system of equations. The lower triangular matrix is represented by a row-oriented sparse data structure. The column of the *j*th nonzero element of the *i*th matrix row is given by `A[i].column[j]`, the *j*th nonzero element of the *i*th row by `A[i].value[j]`, and the number of nonzeros in row *i* is given by `A[i].ncol`. The solution array is represented by the array *y*, the right-hand side of the equation by the array `rhs`, and the number of equations in the system by *N*.

```
for(i=0;i<N;i++)
 {
 y[i] = rhs[i];
   for(j=0;j<A[i].ncol;j++)
   {
   y[i] -=A[i].value[j]*[A[i].column[j]];
   }
 }
```

FIG. 1. *Lower triangular solve using sparse matrix notation.*

We will say that a value of the outer loop index i, i_1 has a dependence on another value of the outer loop index i_2 if the computation of $y[i_1]$ requires $y[i_2]$. The data dependencies between row substitutions indexed by the variable i in 1 are determined by the values assigned during program execution to the data structure A.

When A arises from the zero fill incomplete factorization of a matrix obtained from a partial differential equation discretized over a mesh, the dependency graph between outer loop indices in 1 is related to the undirected graph describing the mesh.

In the directed graph between outer loop indices of (1), a link exists from index i_1 to i_2 when there is a link between i_1 and i_2 in the original mesh and when $i_1 < i_2$. The structure of the directed acyclic graph G produced by the incomplete factorization depends on the ordering of the points of the original mesh. The structure of the sparse data structure A used to represent the DAG depends in turn on the ordering of the points in G.

In an n-by-n mesh with a five-point template, each nonboundary meshpoint $x_{i,j}$ is linked to $x_{i-1,j}\,x_{i+1,j}\,x_{i,j-1}$, and $x_{i,j+1}$. If the meshpoints are ordered in the standard way, i.e., so that the most rapidly varying index is j, the associated DAG G links $x_{i,j}$ to $x_{i+1,j}$ and $x_{i,j+1}$. If we again order the points in the standard way, we obtain a sparse data structure A representing a block lower bidiagonal matrix. The diagonal blocks are n by n lower bidiagonal matrices, and the lower bidiagonal blocks are n by n diagonal matrices. With A defined this way, program 1 represents a two-dimensional recurrence relation that uses only one index i to represent both dimensions. All geometrical information is encoded in the data structure A. Our partitioning strategy takes advantage of the regularity of the underlying geometry, but in partitioning, uses only information encoded in A.

The first step in the preprocessing is to partition the indices of the outer loop of 1 into disjoint sets S_i. All row substitutions in a set S_i can be carried out independently. To obtain the sets S_i, we perform a topological sort of the directed acyclic dependence graph G that describes the dependencies between the outer loop indices of 1. A stage of this sort is performed by removing all indices of G not pointed to by graph edges, then removing all edges that emanated from the removed indices. All indices removed during stage i form the set S_i mentioned above; the elements of S_i are said to belong to *wavefront i*. An adaptation of a common topological sort algorithm [9] allows efficient calculation of the wavefronts of G.

The points of G are then partitioned into a different collection of disjoint sets called *strings*, generated through the following sequence of depth first traversals.

We define a start vertex of G as a vertex not pointed to by any edge. The vertices of S are chosen in the following way. A start vertex V of G is picked, all edges emanating from V are removed; if a new start vertex V' is created through the removal of edges, V' is included in the string. The process is continued recursively to remove as many vertices as possible from G, assigning them to S. When the removal of a vertex exposes multiple start vertices, only one of these start vertices is included in S. As each vertex V' is assigned to S, we mark the vertices W remaining in G that had edges arising from V'. New strings are begun using available start vertices. In picking vertices to incorporate in all strings after the first, priority is given to vertices previously marked by other strings.

Strings have the following properties: (1) The points in each string are connected. (2) There is no more than one point belonging to a given wavefront in a given string. (3) The graph describing the *interstring* dependencies is a directed acyclic graph. The DAG describing the interstring dependencies will be called the *string* DAG.

We will present an example to illustrate how we will use a string decomposition. Figure 2 depicts a DAG that could be obtained from a zero fill incomplete factorization of a matrix arising from the discretization of an elliptic partial differential equation using a nine-point star template. If we perform the topological sort described above, we partition the index set into 16 wavefronts. The work corresponding to each wavefront can be carried out concurrently. Figure 3 depicts the wavefronts assigned to each domain index. Figure 4 depicts a string decomposition and illustrates the string DAG corresponding to this problem. Assume in this running example, we assign two contiguous strings to each of three processors. We allow processor 1 to perform work associated with two consecutive wavefronts in the first string in between synchronizations. In between each pair of synchronizations, all processors are allowed to calculate row substitutions for as many indices as the data dependency relations will allow

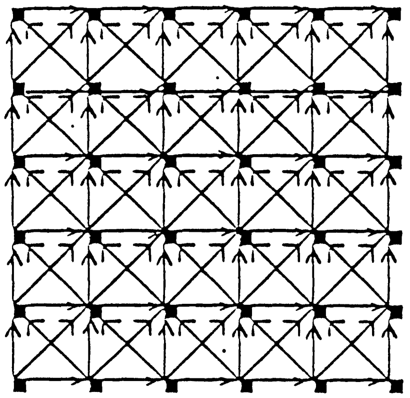


FIG. 2. *Example* DAG.

11	12	13	14	15	16
9	10	11	12	13	14
7	8	9	10	11	12
5	6	7	8	9	10
3	4	5	6	7	8
1	2	3	4	5	6

FIG. 3. *Example* DAG *wavefronts.*

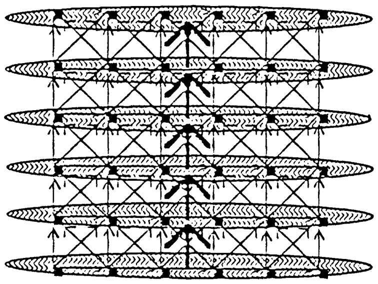


FIG 4. *String* DAG.

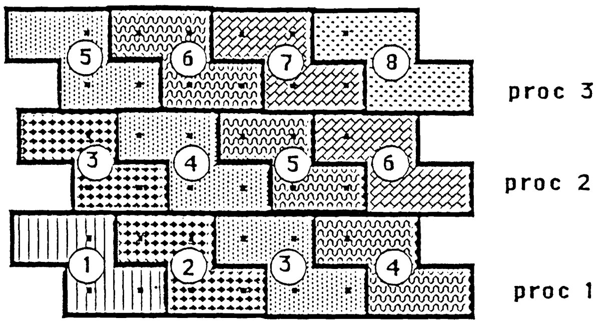


FIG. 5. *New clusters of work.*

(Fig. 5). The number of synchronizations required in this case is only eight, although, as we will see, this reduction in synchronizations may have to be purchased at the cost of a degraded load balance.

On message passing machines, mapping large contiguous sections of the string DAG onto each processor will tend to minimize communication costs, but will also tend to lead to poor load distributions. Scattering or wrapping strings that are contiguous in the string DAG may lead to a much better load distribution at the price of increased communication costs.

In the above example, we can see that with a string decomposition, no global synchronization is actually necessary. Processor 1 needs only to notify processor 2 when it has completed its work, and processor 2 need only notify processor 3. Without some sort of geometrical cues, global synchronizations would clearly be needed.

A problem can generally be partitioned into strings in several different ways. For instance, for the problem in the running example, Fig. 6 depicts another string partitioning. When a DAG originates from the decomposition of a domain, it is possible to make sure that the strings are chosen in a much more controlled manner. The decomposition can be determined by the way in which the meshpoints are ordered in the formation of the matrix. It is simple to arrange for the algorithm to give preference to lower-numbered rows when forming new strings from start vertices D, and to attempt to incorporate rows into a growing string in order of increasing row number.

3. Construction of work schedules. Once we have mapped the strings comprising the string DAG onto processors, we still face the task of deciding when data dependencies allow us to solve for any given unknown. We derive expressions denoting when in a computation we should solve for a row. These expressions take into account only wavefront number and position in the string DAG.

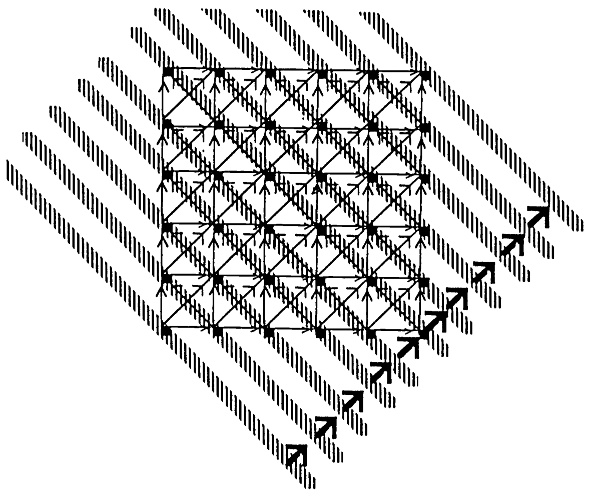


FIG. 6. *Alternate string partition of example problem.*

We will assume that the strings making up the string DAG have been linearly ordered and that contiguous blocks of b strings are demarcated and assigned to consecutively indexed processors in a wrapped manner. In the following, we will say we have computed wavefront q in some block of strings when we compute values for all matrix rows with wavefront q. As we saw in the last section from the running example, we can either use global synchronizations to coordinate work or we can employ information provided by the string DAG to use a form of local synchronization. In local synchronization, processors are logically arranged in a ring; each processor need only synchronize with two logically neighboring processors. In the following, computational work that occurs on a processor between two synchronizations will be referred to as a *phase.*

We will present expressions that give the largest wavefront that the strings in a block must compute during a particular phase. The nature of this schedule depends on (1) the synchronization mechanism used, (2) the data dependency relationships between the strings of the string DAG, and (3) the dependencies between the blocks into which the string DAG is partitioned. Proofs of the correctness of these expressions will be presented in the appendix.

When global synchronization methods are used, all processors must finish phase $p-1$ before any processor is allowed to begin phase p.

The expressions below give the maximum wavefront number that can be computed by a given block i during phase p, assuming the first block computes exactly w wavefronts per phase; i.e., during phase p the first block computes wavefronts $w(p-1)+1$ to wp.

PROPOSITION 1. *Assume that strings making up the string* DAG *have been linearly ordered, that contiguous blocks of strings are demarcated, and that these blocks are assigned to consecutively indexed processors in a wrapped manner. Let W_p^i represent the largest wavefront that can be scheduled during phase p by block i under the following conditions*: (1) *the first block advances w wavefronts per phase, i.e., $W_p^1 = wp$, and* (2) *all required data are computed by each processor before it reaches phase p.*

Then, W_p^i is given by the expression $W_p^i = \max(p, w(p-i+1)+i-1)$.

When it is known that data dependencies occur only between adjacent strings, a more aggressive scheduling policy can be used.

PROPOSITION 2. *Assume that strings making up the string* DAG *have been linearly ordered so that data dependencies occur only between adjacent strings, that contiguous blocks of strings are demarcated, and that these blocks are assigned to consecutively indexed processors in a wrapped manner. Let W_p^i represent the largest wavefront that can be scheduled during phase p by block i under the following conditions*: (1) *the first block advances w wavefronts per phase, i.e., $W_p^1 = wp+b-1$, and* (2) *all required data are computed by each processor before it reaches phase p.*

Then, W_p^i is given by the expression

$$W_p^i = \begin{cases} w(p-i+1)+ib-1 & \text{if } p \geqq i, \\ bp & \text{if } 0 \leqq p < i. \end{cases}$$

The method used for local synchronization requires each processor to interact with only two other processors. Each processor increments an element of a shared array when it finishes each computational phase. The processor to the right in the logical ring of processors is able to know the progress of its left neighbor by reading the left neighbor's shared-memory array element. A processor can carry out work scheduled for phases numbered up to $p+1$, as long as its left neighbor has completed phase p.

The schedule to be presented below makes allowance for the weak interprocessor synchronization by taking into account the data dependencies between blocks of strings in the string DAG.

When local synchronization is used, Proposition 3 below presents expressions that give the maximum wavefront number that is to be computed by a given block i during phase p, assuming that (1) the first block computes exactly w wavefronts per phase, and (2) block i can require data only from blocks j, $\max(i-d, 1) \leqq j < i$. When $d = 1$, we obtain the situation when block $i+1$ requires data only from block i. In this case constraints for scheduling wavefront execution during a given phase p are identical for local and barrier synchronization mechanisms.

The string DAGs that arise from many problems obtained from incomplete factorizations of matrices arising from partial differential equations are frequently characterized by small values of d.

When $d < P$, the expressions below yield the largest wavefront that could be scheduled by a given block at a particular time; when $d \geqq P$, the expressions still allow the data dependencies in the problem to be satisfied, but the expressions no longer necessarily give the largest wavefront that could be scheduled during a given phase.

PROPOSITION 3. *Assume that strings making up the string* DAG *have been linearly ordered, that contiguous blocks of strings are demarcated, and that these blocks are assigned to consecutively indexed processors in a wrapped manner. Assume that each block constitutes a process that executes its computations in phases subject to the constraint that at any time, if block i has finished phase p, block* $i+1$ *can complete all phases with numbers less than or equal to* $p+1$. *Furthermore assume that each block i requires data only from blocks* $\max(i-d, 1)$ *through* $i-1$, *and that* $d < P$.

Let W_p^i *represent the largest wavefront that can be scheduled during phase p by block i under the following conditions.* (1) *The first block advances w wavefronts per phase, i.e.,* $W_p^1 = wp$. (2) *Each processor computes all required data before it reaches phase p.*

Then, for $i \geqq 2$, W_p^i *is given by the expression*

(1) $$W_p^i = \max(\lceil p/d \rceil, w(p-i+1) + \lceil (i-1)/d \rceil).$$

4. Model problem analysis. We will examine load balance synchronization cost tradeoffs in the context of solving a lower triangular system generated by the zero fill factorization of the matrix arising from a X by Y point rectangular mesh with a five-point template. We will utilize P processors and partition the domain into n horizontal strips where each strip is divided into m blocks. We assume for convenience that m and n are multiples of P, and let S be the time required to perform the sequential computation. We define T_B to be the time taken to perform the computation in a block for a given m, n, and S and assume that $T_B = S/mn$.

In a shared-memory machine, estimated total execution time can be expressed as the sum of the time that would be required were the computation evenly distributed between the processors in the absence of any load imbalances, the time wasted due to load imbalances, and the time spent synchronizing.

If it were possible to distribute all work evenly between processors, the computation would require time S/P. The term for the idle time can be derived by noting that during any phase $j \leqq \min(m, n) - 1$ when j is not a multiple of P, there are $P - j \bmod P$ processors idle. When j is a multiple of P, no processors are idle. Thus the sum of the processor idle time for $j \leqq \min(m, n) - 1$ is

$$\frac{T_B \min(m, n) \sum_{l=1}^{P} (l-1)}{P * P} = \frac{T_B \min(m, n)(P-1)}{2P}.$$

Through similar reasoning, the sum of the processor idle time for the last $\min(m, n) - 1$ phases is the same. During the intermediate phases, the load is balanced with $\min(m, n)$ blocks assigned to each processor. Thus the total idle time is

(2) $$\frac{T_B \min(m, n)(P-1)}{P}.$$

If T_{synch} is the cost of a single synchronization, the time required to synchronize is T_{synch} times the number of synchronizations needed, i.e., $T_{synch}(n+m-1)$.

All of the above expressions are symmetric in m and n, i.e., we would get the same predicted efficiency by choosing a block size a and a window size b as we would by choosing a block size b and window size a. We will assume without loss of generality that $m > n$ and rewrite (2) substituting S/mn for T_B, obtaining $S(P-1)/mP$.

When $m > n$, the value chosen for n only effects synchronization costs. When m and n are both multiples of P, the best performance occurs with $n = P$. The window size can be profitably increased to Y/P.

We derive an expression (3) for estimated speedup by dividing the sequential execution time S by an expression for execution time on the multiprocessor obtained by summing the above time estimates, with n set to P:

$$(3) \qquad \frac{1}{\dfrac{P-1}{mP}+\dfrac{1}{P}+\dfrac{T_{synch}(P+m-1)}{S}}.$$

For a fixed number of processors, the speedup obtained as we increase S can depend on how the shape of the problem domain varies as the problem grows, as well as how we change m and n as the domain size varies. When m and n are kept constant as S is increased, regardless of how the shape of the domain changes, the relative cost of load imbalance remains constant, but the relative cost of synchronization decreases. In this case, for large S we obtain an asymptotic speedup of $mP/(P-1+m)$.

If the number of domain points X grows while Y is fixed, we can allow m to grow linearly with X so that the number of points in a window is held constant. In this case, the relative contribution of the time lost due to load imbalance diminishes with increasing S, and one obtains an asymptotic speedup of $P/(1+PT_{synch}/S_0)$ where $S_0 = S/m$.

When X is proportional to Y, an asymptotic speedup equal to the number of processors P can be obtained if we allow m to grow linearly with X.

5. Wavefront longest processing time scheduling. Propositions (1) and (2) describe a parametric method of constructing work schedules when we use global synchronization. These propositions may also be regarded as a way of parametrically describing the wavefronts of a new coarse-grained DAG, each vertex of which represents the solution of a number of rows. Consequently it is natural to consider balancing the processor load for the wavefronts of this coarse-grained DAG, i.e., balancing the load during each phase of computation.

All of the clusters of work executed during a phase of computation are independent. The scheduling of independent tasks to obtain a minimum finishing time is known to be NP-hard. There exist a variety of methods for obtaining approximate solutions to this problem [5], [8], [2]. One method that has been extensively studied is the *Longest Processing Time* or the LPT schedule. A list representing tasks to be scheduled is sorted in descending order of estimated execution time. Consecutive elements of the list are assigned to the processor with the smallest cumulative estimated execution time. The LPT rule requires time $r \log r$ to schedule the execution of r tasks. The performance obtained through the use of the LPT scheduling algorithm is compared with that obtained through the use of a wrapped assignment of strings in the following section.

6. Experimental results.

6.1. Preliminaries. First we will briefly describe the architecture of the Encore Multimax. The Multimax is a bus-based shared-memory machine that utilizes 10 MHz NS32032 processors and NS32081 floating point coprocessors. Processors, shared memory, and i/o interfaces communicate using a 12.5 MHz bus with separate 64-bit data paths and 32-bit address paths.

All tests reported were performed on a machine with 16 processors and 16 Mbytes memory at times when the only active processes were due to the author and to the operating system. On the Encore, the user has no direct control over processor allocation. Tests were performed by spawning a fixed number of processes and keeping the processes in existence for the length of each computation. The processes spawned are scheduled by the operating system, and throughout the following discussions we make the tacit assumption that there is a processor available at all times to execute each process. To reduce the effect of system overhead on our timings, tests were performed using no more than 14 processes.

There does not appear to be significant contention-based performance degradation in programs with the mix of computations and memory references seen here. In a set of experiments using a variety of sparse lower triangular matrices, multiple identical sequential forward solves were run on separate processors at the same time. Timings from these experiments deteriorated by less than 1 percent as the number of processors used was increased from 1 to 14.

6.2. Effect of window and block size on performance. We investigated the effect of window size on execution time under conditions of varying global synchronization cost. The data depicted in Fig. 7 were obtained through a forward solve of the zero fill factorization of a matrix generated using a 100-by-100 point square mesh, in which a five-point template was employed. This matrix is extremely sparse; there are no more than two nonzero off-diagonal elements in any matrix row. Horizontally-oriented strings

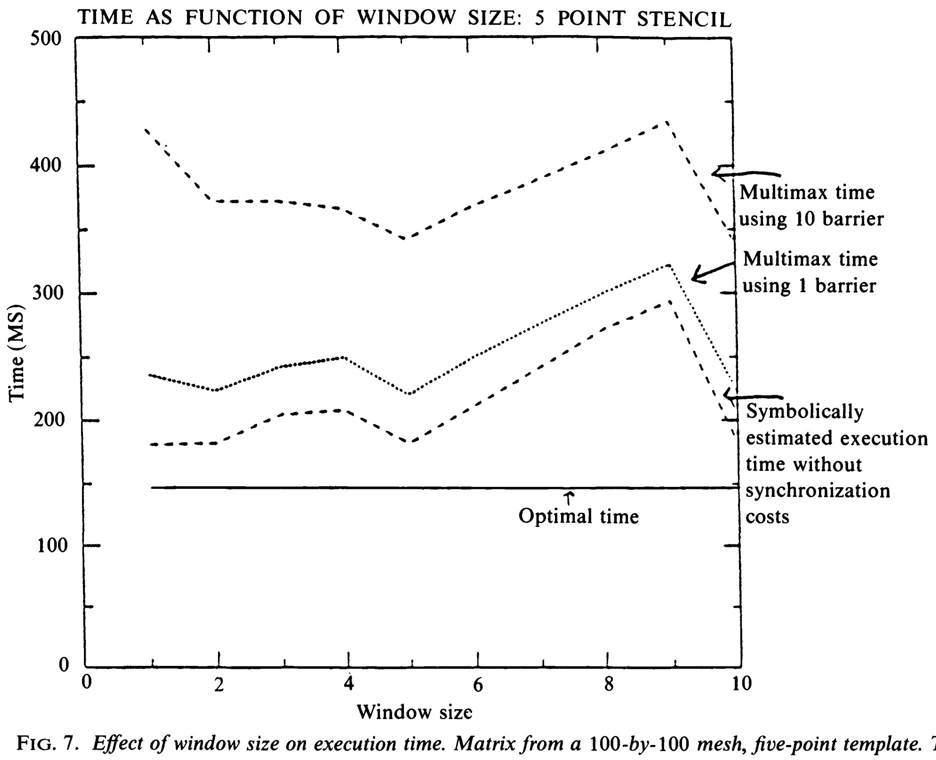


FIG. 7. *Effect of window size on execution time. Matrix from a* 100*-by-*100 *mesh, five-point template. Ten processors used, timings for* 25 *consecutive trials averaged.*

were used in all experimental results reported here unless another orientation is explicitly specified. Ten processors were used to solve this problem, timings were averaged over 25 consecutive runs, and a block size of 1 was employed.

When timed separately, the global synchronization used was found to require 75 microseconds; this compares to approximately 20 microseconds required for a single precision floating point multiply and add. Effects of varying global synchronization costs were simulated by employing either one or ten 75-microsecond global synchronizations between phases.

An estimate was made of the execution time we would expect in the absence of synchronization delays. This uses a program written to give an optimistic estimate for the speedup that could be obtained from a problem given a particular partition of work. This *symbolically estimated speedup* assumes that floating point computations take unit time and that all other computations and synchronizations are instantaneous.

Dividing the execution time of the parallel code running on one processor by the symbolically estimated speedup gives an estimate of what the execution time should be on the basis of load imbalance alone. This time estimate will be referred to below as the *symbolically estimated execution time.* These results are depicted in Fig. 7.

As predicted by the discussion in § 4, the estimate of the execution time in the absence of synchronization costs are virtually identical for window sizes of one, two, five, and ten; when these window sizes are chosen, the horizontal axis of the underlying domain is divided into some multiple of P pieces. The computation times estimated for windows of other sizes reflect the load imbalance caused by the uneven partition of the domain. The experimental data in this figure clearly illustrate that using larger windows becomes more advantageous as synchronization costs increase.

Finally, for the sake of comparison with the experimental results, the time required to solve the problem using the sequential code was divided by the number of processors used; this is called the *optimal time.*

Tradeoffs between load imbalance and synchronization costs were examined in a different manner by plotting the symbolically estimated speedup against the number of phases required to complete a problem. The symbolically estimated speedup is a measure of load balance while the number of phases required to solve a problem is equal to the number of synchronizations that must be performed.

In Fig. 8 the symbolically estimated speedup was plotted against phases required for solving a lower triangular system generated by zero fill factorization of a matrix arising from a 75-by-75 point mesh using a nine-point template.

Estimated speedups are depicted arising from (1) the use of blocks of sizes 1 and 2 where window size varied from 1-8, (2) the use of windows sizes 1 and 2 where the size of blocks varied from 1-8, and (3) the use of a block size equal to the window size where both are varied from 1-6.

The tradeoff between speedup and number of phases appears to be generally more advantageous when large windows and small block sizes are used than when the situation is reversed. The number of phases declines with increasing window and/or block size while the load balance exhibits substantial fluctuations. The tradeoff between load imbalance and number of phases required, appears to be much smoother when the size of the window used is set equal to the size of the block than in the other cases discussed above. With block size equal to window size, the estimated speedup appears to be a gradually decreasing function of the block and window size.

The relative performance of four combinations of window and block size in the face of increasing synchronization costs are depicted in Table 6.1. The execution time required to solve the lower triangular system derived from the 75-by-75 point mesh

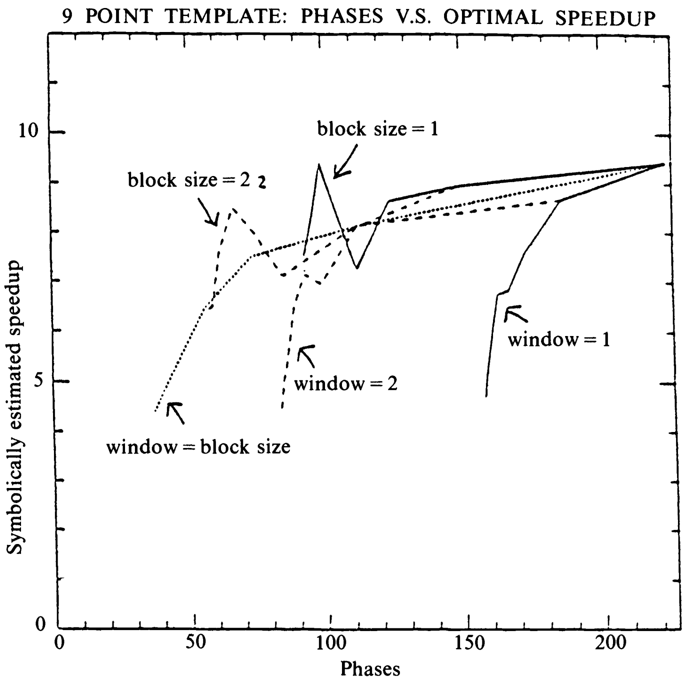


FIG. 8. *Symbolically estimated speedup versus phases required to solve problem on 12 processors. Matrix from a 75-by-75 point mesh, nine-point template. Horizontal strings used.*

TABLE 6.1
Effect of window, block size on execution time. Matrix from a 75-by-75 point mesh, nine-point template, 12 processors, 25 consecutive trials averaged.

Multiples of barrier time	Window 1 block 1	Window 4 block 1	Window 1 block 4	Window 2 block 2
1	0.20	0.23	0.25	0.21
4	0.29	0.27	0.32	0.25
8	0.39	0.32	0.40	0.32
10	0.45	0.35	0.44	0.35

TABLE 6.2
Effect of window, block size on number of phases, and symbolically estimated speedup. Matrix from a 75-by-75 point mesh, nine-point template, 12 processors.

Block size	Window	Phases	Est. speedup
1	1	223	9.43
1	4	112	7.26
4	1	167	6.84
2	2	112	8.14

described above was measured. The following combinations of window and block size were employed: (1) window size = 1, block size = 1; (2) window size = 4, block size = 1; (3) window size = 1, block size = 4; and (4) window size = 2; block size = 2. Between phases, we employed from one to ten 75-microsecond barrier synchronizations. As synchronization costs increase, it becomes more advantageous to reduce the number of phases required to solve a problem even at the cost of increased load imbalances.

The numbers of phases and the symbolically estimated speedup for each of these cases are listed in Table 6.2.

As we can observe from Table 6.2, block size = 4, window size = 1 and block size = 1, window size = 4 require at least as many phases as does block size = 2, window size = 2, and the later achieves a superior load balance. The use of block size 1, window size 1 allows us to achieve a load balance that is even better, but at the cost of added phases of computation. In Table 6.2, for barrier times between 75 and 150 microseconds, the shortest run times were obtained using block size and window size both equal to 1. When barriers were utilized that required more than 150 microseconds, the use of block and window sizes both equal to 2 lead to the shortest run times.

6.3. String orientation effects. The relative merits of using horizontal versus diagonal strings in partitioning a mesh with a nine-point template were investigated. In Fig. 9 is plotted the time required for 12 processors to solve a lower triangular system generated by a zero fill factorization of a matrix arising from a 75-by-75 point mesh. The block size was kept constant at 1, and the window size was varied from 1-8. Tests were carried out using both single 75-microsecond barriers between computational phases and using ten 75-microsecond barriers between phases. For each synchronization cost and window size investigated, the time required for solving the

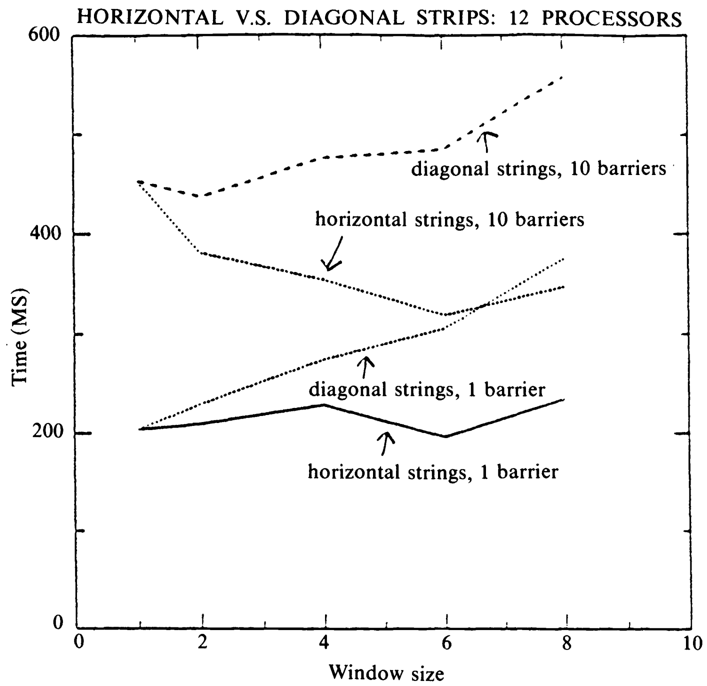


FIG. 9. *Effect of string orientation on execution time. Matrix from a 75-by-75 mesh, nine-point template. Twelve processors, block size = 1, timings for 25 consecutive trials averaged.*

problem using diagonal strings was greater than that required when horizontal strings were used. A substantial reduction in execution time occurred with increasing window size when horizontal strips were employed and interphase synchronization was expensive.

6.4. Comparison between LPT and wrapped scheduling. When barrier synchronization is used, the assignment of blocks to processors plays no role in synchronization, and the work in a computational phase can be freely assigned to processers to optimize the load balance. Experimental comparisons will now be made between the performance that can be achieved through the use of Least Processing Time heuristic and that obtained by assigning the workload to the processors in a wrapped fashion.

The performance difference produced by these scheduling methods is expected to be noticeable only in problems with some degree of irregularity. If during each phase a number of blocks with identical computational requirements had to be executed, a wrapped assignment should lead to an optimal balance of load during that phase.

Even in problems derived from rectangular meshes in which a uniform template was utilized, the computational requirement of blocks computed during a phase are not identical. In such problems, the blocks derived from meshpoints near the boundaries of the domain will generally require smaller amounts of computation than those derived from points further away from the boundaries. Two sets of experiments were performed to compare LPT and wrapped scheduling using one matrix generated from an 80-by-80 point mesh with a five-point template and another matrix generated from the same mesh using a 13-point template. Block and window size were varied and 10 processors were used. The performance obtained using the two scheduling methods were compared using both symbolically estimated speedups and measured run times on the Encore Multimax. The symbolically estimated speedups were not effected by the scheduling mechanism used, and the Multimax run times measured showed minimal differences in no consistent direction.

More substantial differences in run times were noted in problems possessing irregularities that would lead to more substantial differences in the computational requirements of blocks during each phase. The data depicted in Fig. 10 were obtained through a forward solve on 10 processors of the zero fill factorization of a matrix generated using an 80-by-80 point square mesh. Points in mesh rows 1–29 and rows 51–80 employed a five-point template; points in rows 50–80 employed a 13-point template. The block size was set equal to the window size and both were varied from 1–5. The time required to solve the problem was measured when LPT or wrapped scheduling was used to balance load in each block wavefront. These measurements were made when one barrier was used; to simulate the effects of these manipulations on an architecture requiring more expensive synchronization, measurements were also made using 10 barriers. For window sizes of 2 and 3, LPT scheduling led to shorter run times than did wrapped scheduling. When a window of size 1 was used in this problem, no significant difference between the scheduling mechanisms was measured.

When only one barrier is used, there is no advantage to using a window of size greater than 1 in any event. Consequently on the Encore, for this problem, there appears to be nothing to be gained from using either LPT scheduling or from using methods to increase the granularity of parallelism. On architectures where the costs of synchronization are larger compared to the costs of computation, both LPT scheduling and the use of these granularity increasing methods will be advantageous.

A set of problems exhibiting more dramatic load imbalances was also examined. A lower triangular system was produced by the incomplete factorization of a matrix

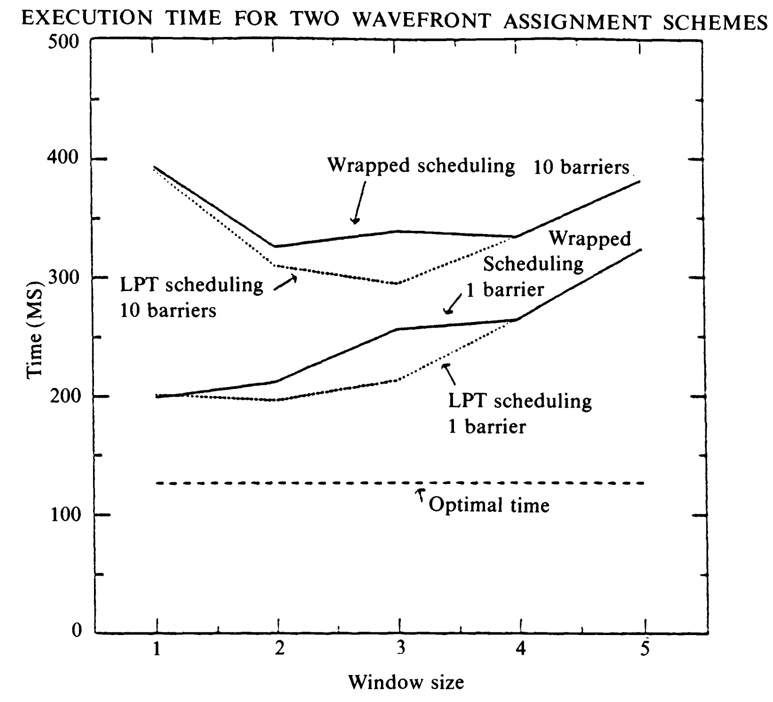


FIG. 10. *Run times from wrapped and* LPT *scheduling. Matrix from* 80-*by*-80 *mesh, mesh rows* 1–29, 51–80: *five-point template*; *rows* 30–50: 13-*point template. Ten processors, block size equal window size, timings for* 25 *trials averaged.*

generated from a 100-by-40 mesh point grid in which the bottom s strips had a 25-point template, and the $40-s$ upper strips had a five-point template. Both the block size and the window size were set equal to 1, and a single barrier was used for synchronization. Figure 11 depicts the efficiency with which 12 processors of the Multimax solves the system as s is varied from 0–12. Efficiency is defined here as the ratio of the time required to solve the problem using a separate sequential code on one processor to the product of the measured time to solve the problem and the number of processors used.

The efficiency obtained through the use of LPT scheduling does not vary much with s, remaining approximately 0.50. The efficiency exhibited by the wrapped scheduling decreases to a low of 0.36 for s equal to 3, but is comparable to the efficiency obtained through the use of LPT when s is close to either 0 or 12. The reasons for this appear to be quite straightforward. When we have, during each phase, a number of very time-consuming blocks that is small compared to the number of processors used, we risk a serious load imbalance when a wrapped assignment strategy is used. As the number of time consuming blocks encountered during each phase increases to approach the number of processors, the amount of wasted processor capacity decreases.

Symbolically estimated execution time calculated for the wrapped assignment added to the synchronization and setup time produce numbers that are quite close to the measured Multimax time for wrapped scheduling. For instance, using 14 processors, the synchronization plus setup time measured separately is 0.03 seconds, the symbolically estimated execution time is 0.13 seconds, and the total time is 0.15 seconds. This correspondence has been noted in the case of the wrapped assignment in a variety of

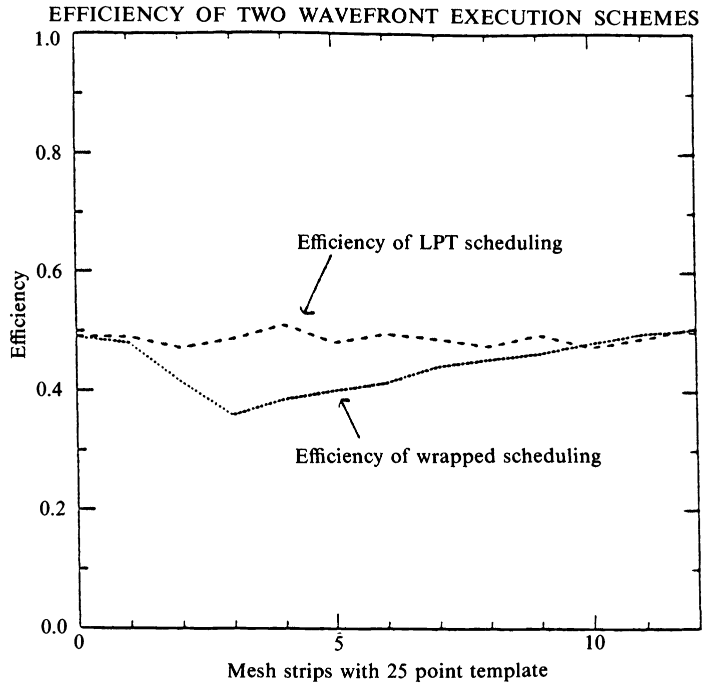


FIG. 11. *Efficiency of wrapped versus* LPT *scheduling. From multimax times. Matrix from* 100*-by-*40 *mesh, bottom strips have* 25*-point template, rest have five-point template. Twelve processors, block, window equal* 1*, timings for* 25 *trials averaged.*

other problems not presented here and gives confidence in the accuracy of the measurements.

When LPT scheduling is used, the symbolically estimated execution time (SEET) appears to have less predictive value. For instance, while the efficiencies obtained through using LPT in Fig. 11 vary little with s, the symbolically estimated execution times vary with s to a substantial extent. For s equal to 2, the measured time taken to solve the problem using the LPT algorithm was 112.16 while the SEET was 69.4; for s equal to 3, the measured time was 118.3 but the SEET was 99.31. The difference in synchronization time between the two cases was, however, quite minimal.

It should be remembered, however, that the LPT scheduling method uses computation times estimates to perform its load balancing. These estimates are not completely accurate. If we measure the SEET obtained after rescheduling computations using a method that produces a reasonable processor schedule based on operation counts, we will tend to obtain overly optimistic estimates of the execution time. This observation points to the obvious importance of accurate run time estimates when performing LPT scheduling.

6.5. A comparison between barrier and local synchronization. While barrier synchronization is relatively inexpensive on the Encore Multimax, local synchronization is less expensive still. As described above, it can be implemented in a way that requires, for each processor, only one shared variable increment followed by a busy wait. Figure 12 depicts a comparison between execution times measured when a barrier was utilized and execution time measured when local synchronization was employed. Both barrier and local synchronization-setup time are also measured and depicted. The problem solved here originates from a 75-by-75 point mesh with a nine-point template; the

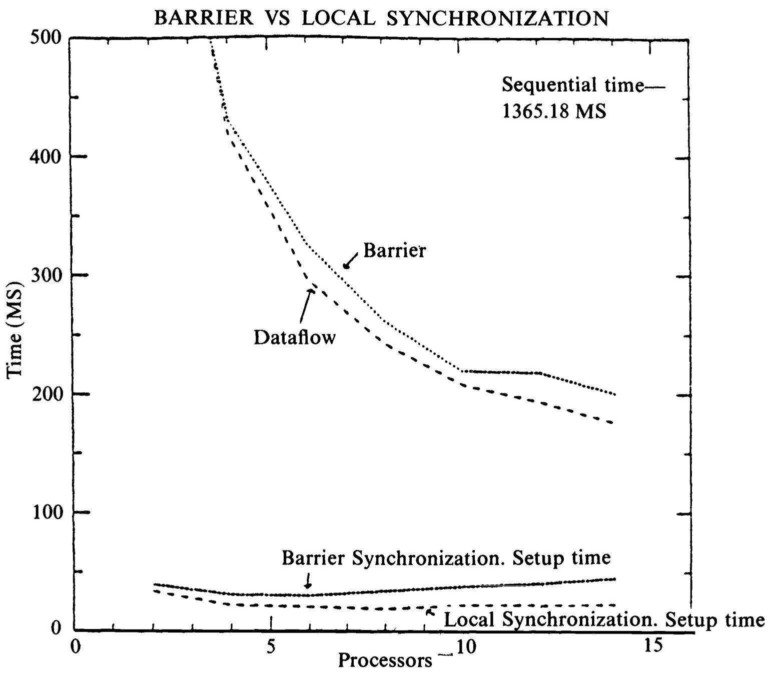


FIG. 12. *Execution time of barrier and local synchronization. Matrix from 75-by-75 mesh, nine-point template. Window, block equal* 1. *Timings from* 25 *consecutive trials averaged.*

window and block size are 1. It is evident from this figure that local synchronization is less expensive than barrier synchronization.

7. Conclusion. In this paper we present a framework for partitioning very sparse triangular systems of linear equations that appears to be flexible enough to produce favorable performance results in a wide variety of parallel architectures. In this paper we have used the Multimax as a hardware simulator to investigate the performance effects of using the partitioning techniques presented here in shared-memory architectures with varying relative synchronization costs.

A method for using the triangular matrix to generate a parameterized assignment of work to processors was described along with simple expressions that describe how to schedule computational work with varying degrees of granularity. These expressions are of considerable practical importance because they allow us to determine easily what computations need to be performed during a given phase to ensure that all data are computed before they are required. The tradeoffs between load imbalance and synchronization costs as a function of block and window size were examined in a variety of contexts. A few comments are in order on which of these techniques we can recommend for use on the current Multimax. On the Encore Multimax, due to its low ratio of synchronization costs to costs of floating point operations, there does not appear to be an advantage in aggregating work to increase the computational granularity. Balancing load within each phase of computation does appear to be advantageous in this architecture, although further practical experience is required to discover when the overhead required for this extra stage of scheduling is worthwhile. The use of local synchronization on the Multimax also appears to be advantageous although its use precludes that of wavefront LPT balancing. We can limit the problem decomposition process to the identification of wavefronts if we have no need to increase granularity through the use of windows or to use strings to implement local synchronization. Hence, on the Encore there should be no reason to pay both the overhead for string decomposition and for LPT balancing.

Appendix.

PROPOSITION 1. *Assume that strings making up the string* DAG *havebeen linearly ordered, that contiguous blocks of strings are demarcated, and that the blocks are assigned to consecutively indexed processors in a wrapped manner. Let* W_p^i *represent the largest wavefront that can be scheduled during phase p by block i under the following conditions.* (1) *The first block advances w wavefronts per phase, i.e.,* $W_p^1 = wp$. (2) *All required data are computed before phase p is reached.*

W_p^i *is given by the expression* $W_p^i = \max(p, w(p-i+1)+i-1)$.

In scheduling work for block i during phase p, we must take into account the numbers of the wavefronts corresponding to the latest available results from blocks $1 \leqq j < i$, since block i may require results from any of these blocks. Since no work can be performed before the first phase, we set $W_p^i = 0$ for $p = 0$. The number of the smallest wavefront corresponding to any result that might be needed by block i at the beginning of phase p may be expressed as

$$\lim_{1 \leqq j < i} W_{p-1}^j.$$

Consequently,

$$W_p^i = \min_{1 \leqq j < i} W_{p-1}^j + 1$$

for $p \geqq 1$.

We now use the above to prove that for all $p \geqq 1$, if $\hat{W}_p^i = \max(p, w(p-i+1)+i-1)$ then $W_p^i = \hat{W}_p^i$. This proof proceeds by induction on block number i.

For $i = 1$, by assumption $W_p^1 = wp$. Since $\hat{W}_p^1 = \max(p, wp)$, $W_p^1 = \hat{W}_p^1$.

We will now use the induction hypothesis for $j \leqq i$ to show $W_p^{i+1} = \hat{W}_p^{i+1}$ for $p \geqq 1$ and $i \geqq 2$. We are assuming that for $j \leqq i$ and $p \geqq 1$,

$$W_p^j = \max(p, w(p-j+1)+j-1).$$

For $p \geqq 2$, $j \leqq i$ we thus have

$$W_{p-1}^j = \max(p-1, w(p-j)+j-1) = \max(p-1, w(p-1)-(w-1)(j-1)).$$

Now

$$W_p^{i+1} = \min_{1 \leqq j < i+1} W_{p-1}^j + 1,$$

so because

$$\min_{1 \leqq j < i+1} W_{p-1}^j = \max(p-1, w(p-i)-(w-1)(i-1)),$$

it follows that

$$W_p^{i+1} = \max(p, w(p-(i+1)+1)+(i+1)-1).$$

Thus $\hat{W}_p^{i+1} = W_p^{i+1}$ and the induction is complete for $p \geqq 2$.

For $p = 1$, since $W_0^j = 0$, $W_1^{i+1} = \min_{1 \leqq j < i+1} W_0^j + 1 = 1$. As it is easily verified that $\hat{W}_1^{i+1} = 1$, $W_1^{i+1} = \hat{W}_1^{i+1}$.

Thus we have shown that for $p \geqq 1$, $W_p^{i+1} = \hat{W}_p^{i+1}$ and the proposition is proved. □

PROPOSITION 2. *Assume that strings making up the string* DAG *have been linearly ordered so that data dependencies occur only between adjacent strings, that contiguous blocks of strings are demarcated, and that these blocks are assigned to consecutively indexed processors in a wrapped manner. Let W_p^i represent the largest wavefront that can be scheduled during phase p by block i under the following conditions.* (1) *The first block advances w wavefronts per phase, i.e., $W_p^1 = wp + b - 1$.* (2) *All required data are computed before the system reaches phase p.*

W_p^i is given by the expression

$$W_p^i = \begin{cases} w(p-i+1)+ib-1 & \text{if } p \geqq i, \\ bp & \text{if } 0 \leqq p < i. \end{cases}$$

Assume that block B has assigned to it strings $v+r$, $1 \leqq r \leqq b$ and that string v has advanced its calculations up to phase p. Due to the nearest neighbor data dependency relations, string $v+r$ may be advanced to wavefront $p+r$. Note that were we not to assume nearest neighbor interstring data dependencies, it is possible that string $v+r$ could have a direct data dependence on string v. In this general case, string $v+r$ could not proceed beyond phase $p+1$. We are thus able to conclude that when we use continuous blocks of b strings each,

$$W_p^i = W_{p-1}^{i-1} + b.$$

Using the above relationship, we will show by induction on block number i that for all $p \geqq 1$, if

$$\hat{W}_p^i = \begin{cases} w(p-i+1)+ib-1 & \text{if } p \geqq i, \\ bp & \text{if } 0 \leqq p < i, \end{cases}$$

then $W_p^i = \hat{W}_p^i$.

For $i=1$, $\hat{W}_p^1 = wp+b-1$ for $p \geqq 1$ so $W_p^1 = \hat{W}_p^1$.

Assume that $W_p^i = \hat{W}_p^i$ for $p \geqq 1$. We will show that $W_q^{i+1} = \hat{W}_q^{i+1}$ for $q \geqq 1$. We first consider the situation that occurs when $q \geqq i+1$. In this case we have

$$W_{p+1}^{i+1} = W_p^i + b = w((p+1)-(i+1)+1)+(i+1)b-1.$$

Since $p+1 \geqq i+1$, the above expression is equal to $\hat{W}_{p+1}^{i+1}$, and consequently $W_q^{i+1} = \hat{W}_q^{i+1}$ for $q \geqq i+1$.

For $0 \leqq p < i$,

$$W_{p+1}^{i+1} = W_p^i + b = b(p+1).$$

Since $p+1 < i+1$, $\hat{W}_{p+1}^{i+1} = b(p+1)$ and hence $W_q^{i+1} = \hat{W}_q^{i+1}$ for $1 \leqq q < i+1$. □

The expression in Proposition 3 is obtained by mapping a chain of logical processes in a wrapped manner onto the P processors of the machine. For the sake of tractability, the expressions are derived under the assumption that the logical processes assigned to a given processor are assumed to be independent of one another. These processes are still subject to the synchronization conditions involving logically neighboring processes, so we always obtain the correct solution to the problem. When $d < P$, we obtain from the proposition the largest wavefront that can be scheduled under the local synchronization assumption; when $d \geqq P$ the expression presented in Proposition 3 yields a wavefront that only guarantees a correct solution under the local synchronization assumptions.

PROPOSITION 3. *Assume that strings making up the string* DAG *have been linearly ordered, that contiguous blocks of strings are demarcated, and that these blocks are assigned to consecutively indexed processors in a wrapped manner. Assume that each block constitutes a process that executes its computations in phases subject to the constraint* *that at any time, if block i has finished phase p, block $i+1$ can complete all phases with numbers less than or equal to $p+1$. Furthermore assume that each block i requires data only from blocks* $\max(i-d, 1)$ *through* $i-1$ *and that* $d<P$.

Let W_p^i represent the largest wavefront that can be scheduled during phase p by block i under the following conditions. (1) *The first block advances w wavefronts per phase, i.e., $W_p^1 = wp$.* (2) *All required data are computed before the system reaches phase p.*

For $i \geqq 2$, W_p^i is given by the expression

$$W_p^i = \max(\lceil p/d \rceil, w(p-i+1) + \lceil (i-1)/d \rceil). \tag{4}$$

In scheduling work for block $i+1$ during phase p, we must take into account the numbers of the wavefronts corresponding to the latest available results from blocks $\max(1, i-d+1) \leqq j < i+1$, since block $i+1$ may require results from any of these blocks. Since no work can be performed before the first phase, we set $W_p^i = 0$ for $p \leqq 0$. The number of the smallest wavefront corresponding to any result that might be needed by block $i+1$ at the beginning of phase p may be expressed for $i \geqq 1$ as

$$\min_{\max(1,i-d+1) \leqq j < i+1} W_{p-i+j-1}^j.$$

Consequently,

$$W_p^{i+1} = \min_{\max(1,i-d+1) \leqq j < i+1} W_{p-i+j-1}^j + 1 \tag{5}$$

for $p \geqq 1$, $i \geqq 1$.

We now use the above to prove by induction on $i \geqq 2$ that for all $p \geqq 1$, if

$$\hat{W}_p^i = \max(\lceil p/d \rceil, w(p-i+1) + \lceil (i-1)/d \rceil), \tag{6}$$

then $W_p^i = \hat{W}_p^i$.

We first establish the base of the induction. By (5), $W_p^2 = W_{p-1}^1 + 1$; as $W_p^1 = wp$ by assumption it follows that $W_p^2 = w(p-1)+1$. From (6), $\hat{W}_p^2 = \max(\lceil p/d \rceil, w(p-1) + \lceil 1/d \rceil)$. For $p \geqq 1$, the above expression is equal to $w(p-1)+1$. Hence $W_p^2 = \hat{W}_p^2$.

Assume that $W_p^j = \hat{W}_p^j$ for $2 \leqq j \leqq i$, $p \geqq 1$, we will show that $W_p^{i+1} = \hat{W}_p^{i+1}$. We first consider the case when

$$p - i + \max(1, i-d+1) - 1 \leqq 0. \tag{7}$$

From (6), for all $1 \leqq j \leqq i$ and $p \geqq 1$, $W_p^j \geqq 1$. Since $W_p^j = 0$ for $p \leqq 0$, from (5) and (7) it follows that $W_p^{i+1} = 1$.

We will show that when (7) is satisfied, it is also the case that $\hat{W}_p^{i+1} = 1$. By (6) $\hat{W}_p^{i+1}$ may be expressed as

$$\hat{W}_p^{i+1} = \max(\lceil p/d \rceil, w(p-i) + \lceil i/d \rceil). \tag{8}$$

When $i \leqq d$, from (7) we have $p \leqq i$ as well as $p \leqq d$. Thus we have $0 < p/d \leqq 1$, $w(p-i) \leqq 0$ and $0 < i/d \leqq 1$ and hence by (8), $\hat{W}_p^{i+1} = 1$.

When $i > d$, from (7), $p \leqq d$. Thus $0 < p/d \leqq 1$ and $\lceil p/d \rceil = 1$. $i > d$ also implies that

$$w(p-i) + \lceil i/d \rceil \leqq w(d-i) + \lceil i/d \rceil$$

and

$$w(d-i) + \lceil i/d \rceil \leqq w(d-i) + i/d + 1.$$

Now $w(d-i) + i/d + 1 \leqq 1$ if and only if $wd(d-i) \leqq -1$. Since $w \geqq 1$, $d \geqq 1$ and $d - i \leqq -1$, we ascertain that $w(p-i) + \lceil i/d \rceil \leqq 1$. Thus when (7) is satisfied, $\hat{W}_p^{i+1} = 1$, and hence in this situation we have shown that $W_p^{i+1} = \hat{W}_p^{i+1}$.

We shall now prove the induction hypothesis when (7) is not satisfied, i.e., when

$$p-i+\max(1, i-d+1)-1 \geqq 1. \tag{9}$$

When $i-d+1 \geqq 2$ we obtain from (5) and (8)

$$W_p^{i+1} = \min_{i-d+1 \leqq j < i+1} \max\left(\left\lceil \frac{p-i+j-1}{d} \right\rceil, w(p-i)+\left\lceil \frac{j-1}{d} \right\rceil\right)+1$$

and hence

$$W_p^{i+1} = \max\left(\left\lceil \frac{p}{d} \right\rceil, w(p-(i+1)-1)+\left\lceil \frac{(i+1)-1}{d} \right\rceil\right).$$

When $i-d+1 \leqq 1$,

$$W_p^{i+1} = \min\left[\min_{2 \leqq j < i+1} \max\left(\left\lceil \frac{p-i+j-1}{d} \right\rceil, w(p-i)+\left\lceil \frac{j-1}{d} \right\rceil\right),\right.$$
$$\left. \max\left(\left\lceil \frac{p-i}{d} \right\rceil, w(p-i)\right)\right]+1;$$

hence

$$W_p^{i+1} = \max\left(\left\lceil \frac{p-i}{d} \right\rceil, w(p-i)\right)+1.$$

By (9), $p-i \geqq 1$ and consequently $w(p-i) \geqq \lceil (p-i)/d \rceil$, and thus

$$\max\left(\left\lceil \frac{p-i}{d} \right\rceil, w(p-i)\right)+1 = w(p-i).$$

Since $1 \leqq i \leqq d$, $\lceil i/d \rceil = 1$ and thus

$$W_p^{i+1} = w(p-i)+\left\lceil \frac{i}{d} \right\rceil = w(p-(i+1)-1)+\left\lceil \frac{(i+1)-1}{d} \right\rceil.$$

We have thus shown that $W_p^{i+1} = \hat{W}_p^{i+1}$ and the proposition is proved. □

Acknowledgments. I wish to thank Stan Eisenstat for making available an inexpensive barrier he has developed on the Multimax and to thank Stan Eisenstat, Dave Nicol, and Martin Schultz for useful discussions. Bob Voigt's careful reading of the manuscript and editorial suggestions are also gratefully acknowledged.

REFERENCES

[1] M. C. Chen, *A design methodology for synthesizing parallel algorithms and architectures*, J. Parallel Distributed Comput., (1986), pp. 116-121.

[2] E. Coffman, M. Garey, and D. Johnson, *An application of bin-packing to multiprocessor scheduling*, SIAM J. Comput., 7 (1978), pp. 1-17.

[3] S. C. Eisenstat, M. T. Heath, C. S. Henkel, and C. H. Romine, *Modified cyclic algorithms for solving triangular systems on distributed-memory multiprocessors*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 589-600.

[4] A. George, M. T. Heath, J. Liu, and E. Ng, *Solution of sparse positive definite systems on a shared-memory multiprocessor*, Tech. Report ORNL/TM-10260, Oak Ridge National Laboratory, Oak Ridge, TN, January 1987.

[5] R. Graham, *Bounds on multiprocessor timing anomalies*, SIAM J. Appl. Math., 17 (1969), pp. 416-429.

[6] A. GREENBAUM, *Solving sparse triangular linear systems using Fortran with parallel extensions on the* NYU *ultracomputer prototype*, Report 99, NYU Ultracomputer Note, New York University, New York, April 1986.

[7] M. T. HEATH AND C. H. ROMINE, *Parallel solution of triangular systems on distributed-memory multiprocessors*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 558-588.

[8] E. HOROWITZ AND S. SAHNI, *Fundamentals of Computer Algorithms*, Computer Science Press, Rockville, MD, 1978.

[9] ———, *Fundamentals of Data Structures*, Computer Science Press, Rockville, MD, 1983.

[10] G. LI AND T. F. COLEMAN, *A parallel triangular solver for a distributed-memory multiprocessor*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 485-502.

[11] R. MIRCHANDANEY, J. SALTZ, R. SMITH, D. NICOL, AND K. CROWLEY, *Principles of runtime support for parallel processors*, in Proc. 1988 International Conference on Supercomputing, St. Malo, France, July 1988, pp. 140-152.

[12] Y. SAAD AND M. H. SCHULTZ, *Topological properties of hypercubes*, YALEU/DCS/RR-389, Department of Computer Science, Yale University, New Haven, CT, June 1986.

[13] J. H. SALTZ, V. H. NAIK, AND D. M. NICOL, *Reduction of the effects of the communication delays in scientific algorithms on message passing* MIMD *architectures*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s118-s134.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 1, pp. 145–160, January 1990

ON THE COMPLEXITY OF SPARSE GAUSSIAN ELIMINATION VIA BORDERING*

RANDOLPH E. BANK† AND DONALD J. ROSE‡

Abstract. The complexity of a general sparse Gaussian elimination algorithm based on the bordering algorithm is analyzed. It has been shown that this procedure requires less integer overhead storage than more traditional general sparse procedures, but the complexity of the nonnumerical overhead calculations was not clear. Here the nonnumerical complexity of the original procedure is shown to be comparable to the numerical complexity for an $n \times n$ grid graph, and an enhancement of the procedure that can reduce the overhead is presented.

Key words. sparse Gaussian elimination, bordering, m-tree

AMS(MOS) subject classifications. 65F05, 65N20

1. Introduction.

In this paper, we consider the solution of the $N \times N$ linear system

$$Ax = b \tag{1}$$

where A is large, sparse, symmetric, and positive definite. We consider the direct solution of (1) by means of general sparse Gaussian elimination. In such a procedure, we find a permutation matrix P, and compute the decomposition

$$PAP^t = LDL^t$$

where L is unit lower triangular and D is diagonal. The system (1) is then solved by

$$\begin{aligned} Lw &= Pb, \\ Dy &= w, \\ L^t z &= y, \\ x &= P^t z. \end{aligned}$$

Several good ordering algorithms (nested dissection and minimum degree) are available for computing P [5], [9]. Since our interest here does not focus directly on the ordering, we assume for convenience that $P = I$, or that A has been preordered to reflect an appropriate choice of P.

Our purpose here is to examine the nonnumerical complexity of the sparse elimination algorithm given in [3]. As was shown there, a general sparse elimination scheme based on the bordering algorithm requires less storage for pointers and row/column indices than more traditional implementations of general sparse elimination. This is accomplished by exploiting the m-tree, a particular spanning tree for the graph of the filled-in matrix. To our knowledge, the m-tree previously has not been applied in this fashion to the numerical factorization, but it has been used, directly or indirectly, in

* Received by the editors September 23, 1987; accepted for publication November 15, 1988.

† Department of Mathematics, University of California at San Diego, La Jolla, California 92093. The work of this author was supported by the Office of Naval Research under contract N00014-82K-0197.

‡ Department of Computer Science, Duke University , Durham, North Carolina 27706. The work of this author was supported by the Office of Naval Research under contract N00014-85K-0487, AT&T Bell Laboratories, and the Microelectronics Center of North Carolina.

several optimal order algorithms for computing the fill-in during the symbolic factorization phase [4] - [8], [10], [12].

In §2, we review the bordering algorithm, and introduce the sorting and intersection problems that arise in the sparse formulation of the algorithm. In §3, we introduce m-trees (or elimination trees) and review their role in sparse Gaussian elimination. We do not attempt to present an overview here, but rather attempt to focus on those results that are relevant to our particular algorithm. This section assumes prior knowledge of the role of graph theory in sparse Gaussian elimination; surveys of this role are available in [9] and [5]. More general discussions of elimination trees are given in [6] - [8], [12].

In §4, we return to the sorting and intersection problems, and show how m-trees can be exploited effectively in their solution. The sorting problem is relatively straightforward, and its computational complexity is of lower order than the complexity of the numerical factorization. On the other hand, the complexity of the intersection problem is potentially of the same order as the numerical factorization; indeed, in our first formulation of the problem, it becomes clear that the complexity must be at least as great as the numerical factorization. Later we split the intersection problem into two parts, with one corresponding exactly to the numerical factorization, and the second being pure overhead. We then present a new procedure for reducing the complexity of this second part of the intersection problem; this procedure again exploits the structure of the m-tree.

In §5, we analyze the complexity of the old and new approaches to the intersection problem for the special case of an $n \times n$ grid ordered by nested dissection. The special structure of this problem allows us to make exact estimates of the complexity. For the old approach, we show that the complexity of the intersection problem is $O(n^3)$, the same as the complexity of the numerical computations [5], [11]. For the new approach, the complexity of the second part is reduced to $O(n^2(\log n)^2)$. In §6, we touch briefly on the issues of data structures and implementation.

We emphasize that in terms of a practical computer code for doing sparse Gaussian elimination, the best we realistically can expect to achieve for a package based on bordering is an execution time comparable to the better row-oriented general sparse matrix packages currently available (e.g., Yale Sparse Matrix Package [4] and Sparspak [5]), at least for sequential computation. Certainly the number of floating point computations in a general sparse code depends only on the ordering of the equations and the zero-nonzero structure of the original matrix, and this is the same for all procedures. The differences between algorithms are mainly in the ordering of the computations, data structures, and nonnumerical overhead. Here the bordering approach can offer some advantages. It usually requires less integer overhead storage [3] than row schemes, and since the storage required is not a function of the fill-in, the amount of integer overhead is known before the computation begins. Also, some sparse matrix problems present themselves in a way such that a columnwise sparse storage scheme coupled with the bordering algorithm for Gaussian elimination becomes the most convenient and obvious approach to their solution. Indeed, one such application (to the linear systems arising in the hierarchical basis multigrid method [1], [2]) motivated our original exploration of such algorithms.

In terms of nonnumerical computations, our new approach to the intersection problem reduces nonnumerical computations in the numerical factorization phase to a level approximately equal to that of row-oriented schemes, that is, about one indirect address for each floating point multiplication operation in the inner loop in the symmetric case. For nonsymmetric problems with symmetric zero-nonzero patterns, the nonnumerical costs of our bordering approach remain the same, but the number of floating point operations approximately doubles. The number of nonnumerical computations in the forward/backward solution phases has always been about the same for the row- and column-oriented schemes. Thus we need not sacrifice execution time if it seems desirable to use a column-oriented approach.

2. The bordering algorithm and sparse elimination. Let A be a symmetric, positive definite matrix. We consider the factorization

$$A = LDL^t \tag{2}$$

where D is diagonal and L is unit lower triangular. Let A_k denote the $k \times k$ upper left principal submatrix of A, and we assume that we have already computed

$$A_{k-1} = L_{k-1}D_{k-1}L^t_{k-1}$$

by the bordering algorithm. Then

$$\begin{aligned} A_k &= \begin{bmatrix} A_{k-1} & c \\ c^t & \alpha \end{bmatrix} \\ &= L_k D_k L_k^t \\ &= \begin{bmatrix} L_{k-1} & 0 \\ \ell^t & 1 \end{bmatrix} \begin{bmatrix} D_{k-1} & 0 \\ 0 & \delta \end{bmatrix} \begin{bmatrix} L^t_{k-1} & \ell \\ 0 & 1 \end{bmatrix} \end{aligned}$$

where

$$\begin{aligned} L_{k-1}D_{k-1}\ell &= c, \\ \delta &= \alpha - \ell^t D_{k-1}\ell. \end{aligned}$$

Thus, at the kth stage, the bordering algorithm consists of solving the lower triangular system

$$L_{k-1}v = c \tag{3}$$

and setting

$$\ell = D^{-1}_{k-1}v, \tag{4}$$
$$\delta = \alpha - \ell^t v. \tag{5}$$

Elementwise, the algorithm may be written in the following manner:

Procedure Dense Factor.

```
(D1)    for $k = 1, N$
(D2)        $d_{kk} \leftarrow a_{kk}$
(D3)        for $j = 1, k-1$
(D4)            $v_j = a_{jk} - \sum_{i=1}^{j-1} \ell_{ji} v_i$
(D5)            $\ell_{kj} = v_j / d_{jj}$
(D6)            $d_{kk} \leftarrow d_{kk} - \ell_{kj} v_j$
```

Let $\mathcal{C}_k$ denote the index set of nonzeros in column k of $L^t - I$ (row k of $L - I$). Then, for sparse matrices, the factorization algorithm may be written as follows:

Procedure Sparse Factor.

```
(S1)    for $k = 1, N$
(S2)        $d_{kk} \leftarrow a_{kk}$
(S3)        for $j \in \mathcal{C}_k$
(S4)            $v_j = a_{jk} - \sum_{i \in \mathcal{C}_k \cap \mathcal{C}_j} \ell_{ji} v_i$
(S5)            $\ell_{kj} = v_j / d_{jj}$
(S6)            $d_{kk} \leftarrow d_{kk} - \ell_{kj} v_j$
```

Because of the implicit nature of line (S4), the indices j on line (S3) must be sorted such that the right-hand side of line (S4) is always well defined. Sorting $\mathcal{C}_k$ by increasing order is certainly sufficient, but other orderings are possible and will prove more convenient. In particular, any sorting of the indices that allows (3) to be backsolved is acceptable. We will refer to this as the *sorting problem.*

The computation of v_j in line (S4) requires the computation of $\mathcal{C}_k \cap \mathcal{C}_j$. We will refer to this as the *intersection problem.* Since this is an inner loop computation, it clearly contributes to the highest order term of the overall complexity; thus it is important to compute these intersections as efficiently as possible. We will analyze these problems in §§4 and 5.

3. m-trees and sparse elimination. Let $\mathcal{G} = (\mathcal{X}, \mathcal{E}, \alpha)$ be the connected, ordered graph associated with the irreducible, symmetric, positive definite matrix $A = LDL^t$. Here $\mathcal{X} = \{x_i\}_{i=1}^N$ denotes the vertex set, $\mathcal{E}$ the edge set ($e_{ji} = e_{ij} \in \mathcal{E}, i \neq j$ if and only if $a_{ij} \neq 0$), and α is the ordering ($\alpha(i) = x_i$). Let $\mathcal{G}' = (\mathcal{X}, \mathcal{E} \cup \mathcal{F}, \alpha)$ denote the chordal graph generated by α. $\mathcal{F}$ denotes the set of fill-in edges generated by the elimination process.

For $x_i \in \mathcal{X}$,

$$\operatorname{adj}(x_i) = \{y_j | e_{ij} \in \mathcal{E} \cup \mathcal{F}\}$$

denotes the adjacency of x_i in $\mathcal{G}'$. We denote the monotone adjacency of x_i by

$$\operatorname{madj}(x_i) = \{y_j \in \operatorname{adj}(x_i) | j > i\}$$

and set

$$\operatorname{cadj}(x_i) = \operatorname{adj}(x_i) - \operatorname{madj}(x_i).$$

The index set associated with $\operatorname{madj}(x_i)$ is the set of column indices for row i of $L^t - I$, while $\operatorname{cadj}(x_i)$ corresponds to $\mathcal{C}_i$. Additionally, recall that $\operatorname{madj}(x_i)$ is a clique in $\mathcal{G}'$.

Let $\mathcal{G}' = (\mathcal{X}, \mathcal{E} \cup \mathcal{F}, \alpha)$ be a chordal graph and let $m(i), 1 \leq i \leq N-1$ be given by

$$m(i) = \min\{j | x_j \in \operatorname{madj}(x_i)\}. \tag{6}$$

Then the *m-tree* $\mathcal{T}$ for $\mathcal{G}'$ is the tree with vertex set $\mathcal{X}$ and edges $\mathcal{E}' = \{e_{im(i)}\}_{i=1}^{N-1}$. The m-tree is also called the *elimination tree* by Liu [7] and Schreiber [12]. See Liu [8] for a recent survey of the role of m-trees in sparse Gaussian elimination. Among

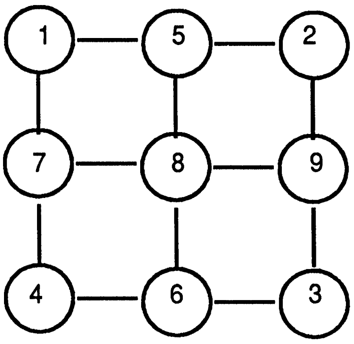


FIG. 1. *A* 3 × 3 *grid graph with nested dissection ordering.*

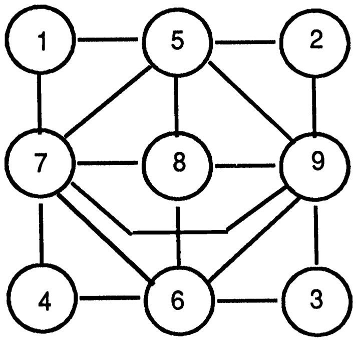


FIG. 2. *Fill-in for the* 3 × 3 *grid graph.*

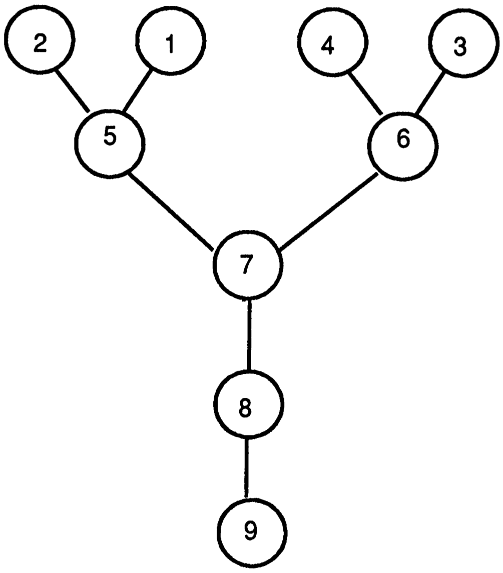


FIG. 3. *An m-tree for the* 3 × 3 *grid graph.*

the m-tree's more important applications is its use in optimal order procedures for computing the fill-in $\mathcal{E} \cup \mathcal{F}$ using the ideas of Rose, Tarjan, and Lueker [10].

As an example, consider the 3×3 grid graph with the nested dissection ordering α, illustrated in Figs. 1 and 2.

For this graph $m(i)$ is defined as follows:

i	1	2	3	4	5	6	7	8	9
$m(i)$	5	5	6	6	7	7	8	9	–

The m-tree is shown in Fig. 3.

LEMMA 3.1. *Let $\mathcal{G}' = (\mathcal{X}, \mathcal{E} \cup \mathcal{F}, \alpha)$ be chordal and let $e_{ij} \in \mathcal{E} \cup \mathcal{F}, i < j$. Then either $m(i) = j$ or $e_{m(i)j} \in \mathcal{E} \cup \mathcal{F}$.*

Proof. See Schreiber [12]. We give proofs of this and other lemmas in this section because of their brevity. If $m(i) = j$, we are done, so we assume that $m(i) = k < j$. Then, since $\text{madj}(x_i)$ is a clique and $x_j, x_k \in \text{madj}(x_i)$, $e_{kj} \in \mathcal{E} \cup \mathcal{F}$. □

LEMMA 3.2. *Let $\mathcal{T}_i$ be the subgraph of $\mathcal{T}$ induced by the set $\{x_i\} \cup \text{cadj}(x_i)$. Then $\mathcal{T}_i$ is connected (i.e., it is a subtree).*

Proof. See Schreiber [12]. We will show the path from $x_j \in \text{cadj}(x_i)$ to x_i in $\mathcal{T}$ contains only vertices in the set $\{x_i\} \cup \text{cadj}(x_i)$. This is done by induction on ℓ, the length of the path. If $\ell = 1$, then $m(j) = i$, and $e_{ij} \in \mathcal{E}'$. We assume the lemma is true for paths of length $\ell - 1$, and show it for a path of length ℓ. Let the path from $x_j \in \text{cadj}(x_i)$ to x_i in $\mathcal{T}$ be of length ℓ, and let $k = m(j) < i$. Note that $e_{ki} \in \mathcal{E} \cup \mathcal{F}$. Thus, $x_k \in \text{cadj}(x_i)$ and the length of the path from x_k to x_i in $\mathcal{T}$ is $\ell - 1$. □

The subtrees for the example 3×3 grid graph are shown in Fig. 4.

Let $\mathcal{L}_i = \{x_j \in \text{cadj}(x_i) | x_j \text{ is a leaf of } \mathcal{T}_i\}$ denote the set of leaves of $\mathcal{T}_i$. For $x_j \in \mathcal{L}_i$, $m(j) \neq i$, let $\{x_{\ell_1}, x_{\ell_2}, \cdots, x_{\ell_k}\}$ be the path of length $k - 1 \geq 2$ from $x_j = x_{\ell_1}$ to $x_i = x_{\ell_k}$ in $\mathcal{T}$. The edge e_{ij} is called a *backedge.* Since $x_{\ell_p} \in \text{cadj}(x_i), 2 \leq p \leq k-1$, $e_{\ell_p i} \in \mathcal{E} \cup \mathcal{F}$. Thus the path and the backedge form a cycle in $\mathcal{G}'$; this cycle is chorded by the edges $e_{\ell_p i}, 2 \leq p \leq k - 1$. Let $\mathcal{B}$ denote the set of backedges and $\mathcal{K}$ the set of chords. It is easy to see

$$\mathcal{E} \cup \mathcal{F} = \mathcal{E}' \cup \mathcal{B} \cup \mathcal{K}.$$

Following Liu [7], the graph $\mathcal{S} = (\mathcal{X}, \mathcal{E}' \cup \mathcal{B})$ is called the *skeleton* of $\mathcal{G}'$. The skeleton of the 3×3 grid graph is shown in Fig. 5.

LEMMA 3.3. *Let $\mathcal{G}$, $\mathcal{G}'$, $\mathcal{E} \cup \mathcal{F}$, and $\mathcal{B}$ be defined as above. Then $\mathcal{B} \subset \mathcal{E}$.*

Proof. See Liu [7] and Schreiber [12]. Let $e_{ij} \in \mathcal{B}, j < i$. Then $x_j \in \mathcal{L}_i$. Suppose $e_{ij} \notin \mathcal{E}$, so $e_{ij} \in \mathcal{F}$. Define

$$k = \max\{p | x_i, x_j \in \text{madj}(x_p)\}.$$

Clearly k exists; otherwise, e_{ij} would not be in $\mathcal{F}$. We now observe that $m(k) = j$; if $\ell = m(k) < j$, then $x_\ell, x_i, x_j \in \text{madj}(x_k)$. Since $\text{madj}(x_k)$ is a clique, $x_i, x_j \in \text{madj}(x_\ell)$, contradicting the definition of k. However, since $m(k) = j$, $e_{kj} \in \mathcal{E}'$, contradicting $x_j \in \mathcal{L}_i$. □

Since $\mathcal{B} \subseteq \mathcal{E}$, the index sets corresponding to the $\mathcal{L}_i$ are subsets of the row indices for the upper triangular part of column i of the matrix A.

4. The sorting and intersection problems. It is apparent that the generation of the sets $\mathcal{C}_k$ required for the numerical factorization requires only knowledge of the sets $\mathcal{L}_k \subseteq \mathcal{C}_k$ and the m-tree for $\mathcal{G}'$. The m-tree $\mathcal{T}$, along with the sets $\mathcal{C}_k$ for all k, can be computed in $O(|\mathcal{E} \cup \mathcal{F}|)$ time using the procedures in [3]. The $\mathcal{C}_k$ need not be

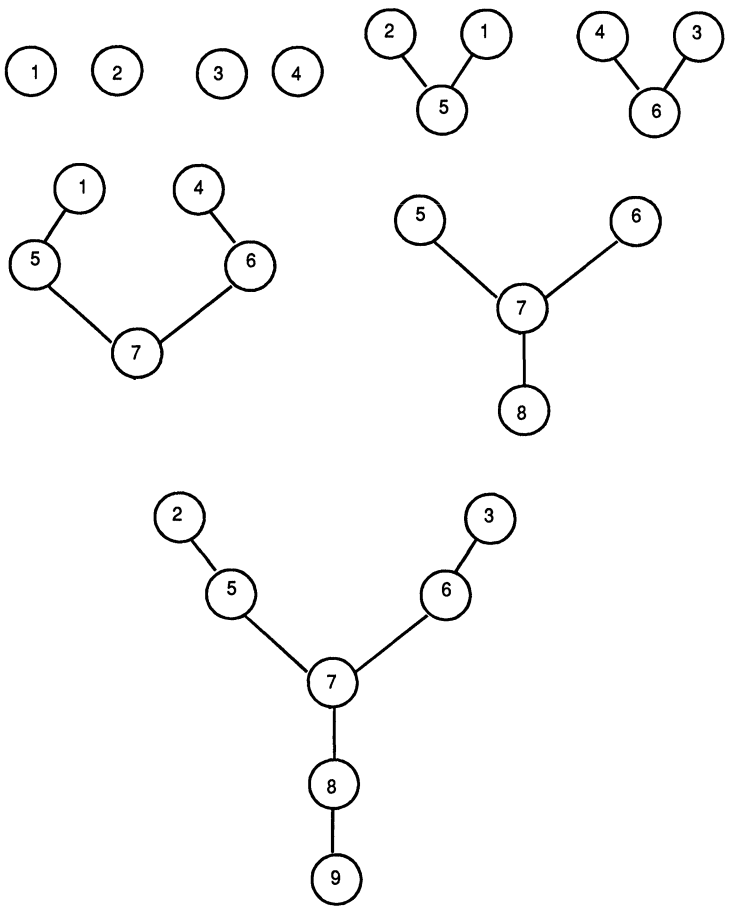


FIG. 4. *Subtrees for the* 3×3 *grid graph.*

permanently stored, since they can be regenerated as needed using the $\mathcal{L}_k$ and the m-tree.

By Lemma 3.3, the index set for $\mathcal{L}_k$ is a subset of the row indices for the nonzeros in the strict upper triangular part of column k of A. It is thus convenient to store the strict upper triangular part of A column by column, since this also facilitates the use of the bordering algorithm. It is not essential that the index set $\mathcal{L}_k$ be explicitly determined; indeed, it is convenient to define *generalized leaves* $\mathcal{L}'_k$ by

$$\mathcal{L}'_k = \{x_j \in \mathcal{C}_k | e_{kj} \in \mathcal{E}, j < k\}.$$

The index set for $\mathcal{L}'_k$ corresponds exactly to the row indices for the nonzeros in the strict upper triangle of column k of A; clearly $\mathcal{L}_k \subseteq \mathcal{L}'_k$.

We next partition the index set $\mathcal{C}_i$ among the generalized leaves for $\mathcal{C}_i$. Thus we let

$$\begin{aligned} \mathcal{C}_i &= \bigcup_{x_j \in \mathcal{L}'_i} \mathcal{D}_{ij}, \\ \mathcal{D}_{ij} \cap \mathcal{D}_{ik} &= \emptyset, \qquad j \neq k. \end{aligned}$$

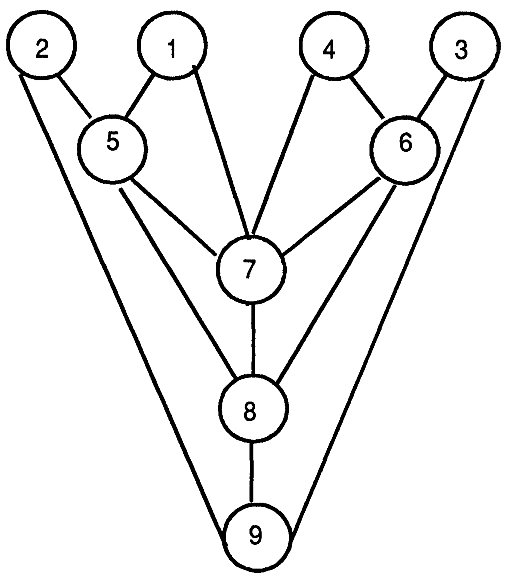


FIG. 5. *Skeleton for the* 3 × 3 *grid graph.*

The sets $\mathcal{D}_{ij}$ are defined as follows: for $x_j \in \mathcal{L}'_i$, $\mathcal{D}_{ij}$ is the index set of vertices on the path from x_j to x_i in $\mathcal{T}$ which do not coincide with the path of any higher ordered vertex in $\mathcal{L}'_i$. For example, in our 3 × 3 grid graph, we have for x_9,

$$\begin{aligned} \mathcal{L}_i &= \{x_3, x_2\}, \\ \mathcal{L}'_i &= \{x_8, x_3, x_2\}, \\ \mathcal{D}_{98} &= \{8\}, \\ \mathcal{D}_{93} &= \{3, 6, 7\}, \\ \mathcal{D}_{92} &= \{2, 5\}. \end{aligned}$$

This partitioning of $\mathcal{C}_i$ results naturally if the vertices in $\mathcal{L}'_i$ are sorted by index and $\mathcal{C}_i$ is generated by processing $x_j \in \mathcal{L}'_i$ in *decreasing* order. Since $|\mathcal{L}'_i|$ is typically not large, sorting these sets as an initialization step generally does not contribute to the highest order complexity terms. Thus we assume that the sorted $\mathcal{L}'_i$ are available as input.

For any $x_j \in \mathcal{L}'_i$, the set $\mathcal{D}_{ij}$ is generated from j and $\ell = |\mathcal{D}_{ij}|$ using the m-function,

$$\mathcal{D}_{ij} = \{j, m(j), m(m(j)), \cdots, m^{\ell-1}(j)\}. \tag{7}$$

We now return to the two problems mentioned at the conclusion of §2. We consider first the sorting problem for $\mathcal{C}_k$ in line (S3) of Procedure Sparse Factor. In light of the analysis of §3, we must sort the vertices in cadj(x_k), which together with x_k are the vertices of $\mathcal{T}_k$, such that the predecessors of vertex $x_j \in$ cadj(x_k) are ordered before x_j itself. If this is done, the right-hand side of line (S4) of Procedure Sparse Factor always will be well defined. One such sorting can be generated easily by processing the vertices in $\mathcal{L}'_k$ in *increasing* order, generating the sets $\mathcal{D}_{kj}$ using the m-function. For example, for vertex x_9 of our 3 × 3 grid graph, this would result in the ordering

$$\begin{aligned} \mathcal{C}_9 &= \{2,5,3,6,7,8\} \\ &= \mathcal{D}_{92} \cup \mathcal{D}_{93} \cup \mathcal{D}_{98}. \end{aligned}$$

Another solution to the sorting problem, based on a renumbering of the vertices using a postorder traversal of the m-tree, is given by Liu [7].

As each element $j \in \mathcal{C}_k$ is generated, we can mark an integer vector c, initialized to zero, to mark the set $\mathcal{C}_k$. It is thus easy to test if $i \in \mathcal{C}_k$ for any i by checking if $c(i) \neq 0$. We assume the existence of such an array as we analyze the intersection problem.

Given $\mathcal{C}_k$, represented by the array c, the problem of computing $\mathcal{C}_i \cap \mathcal{C}_k$ can be done in $O(|\mathcal{C}_i|)$ time by generating

$$\mathcal{C}_i = \bigcup_{j \in \mathcal{L}'_i} \mathcal{D}_{ij}$$

and testing using c. Let $\{\ell_p\}_{p=1}^{|\mathcal{D}_{ij}|}$ be the ordered sequence of indices in $\mathcal{D}_{ij}$. Then, assuming $\mathcal{D}_{ij} \cap \mathcal{C}_k \neq \emptyset$, there will exist a $\bar{p}$ such that

(8) $$\ell_p \notin \mathcal{C}_k, \qquad 1 \leq p \leq \bar{p} - 1,$$

(9) $$\ell_p \in \mathcal{C}_k, \qquad \bar{p} \leq p \leq |\mathcal{D}_{ij}|.$$

This is true since $\mathcal{T}_k$ is a connected subtree of $\mathcal{T}$ and the sequence $\{\ell_p\}$ corresponds to a path in $\mathcal{T}$ generated using the m-function.

Clearly, the second part of the sequence (9) generates actual floating point computations on line (S4) of Procedure Sparse Factor, and thus the complexity of this portion is unavoidably of the same order as the floating point work. On the other hand, generating the first part of the sequence is nonnumerical overhead which does not correspond to anything useful in terms of the numerical factorization.

Since the computation of intersections must contribute to the highest order complexity term, we are interested in finding a procedure for reducing the wasted computuation in (8). We are thus led to define, for each $\mathcal{D}_{ij}$, $i \in \mathcal{C}_k$, the index

(10) $$q_{ijk} = \left\{ \begin{array}{ll} \ell_{\bar{p}} & \mathcal{D}_{ij} \cap \mathcal{C}_k \neq \emptyset \\ 0 & \mathcal{D}_{ij} \cap \mathcal{C}_k = \emptyset \end{array} \right\}$$

for $j < i < k$. We then recast the intersection problem in terms of computing q_{ijk} as quickly as possible, and then view the complexity of the intersection problem in terms of the complexity of computing q_{ijk}.

Let $s \leq t < r$ be integers; we now define the *run* $R = R(r,s) = \{t\}_{t=s}^{r-1}$ such that

$$m(t) = t + 1 \qquad s \leq t \leq r - 1$$

(11) $$m(r) \neq r + 1 \qquad \text{if } r \neq N$$

$$m(s-1) \neq s \qquad \text{if } s \neq 1.$$

Note that a run must contain at least one vertex (in which case $R(r,s) = \{s\}$, $m(s) = s+1$).

It is well known that sparse Gaussian elimination tends to produce large cliques in $\mathcal{G}'$; most of the vertices in these cliques will have $m(k) = k+1$. On the other hand, the formation of a large clique is not a necessary condition for the formation of a large run; for example, $m(k) = k+1$ for all $k < N$ in a tridiagonal matrix.

In any event, because of the restrictions on $m(r)$ and $m(s-1)$ in (11), a given integer t can be in at most one run. Thus we are able to define an *express vector* $e(t)$ which allows us to examine all the vertices in a run in $O(1)$ work. The express vector e, of length N is given by

$$e(t)\begin{cases} = r & \text{if } t \in R(r,s) \text{ for some } r \text{ and } s, \\ \le 0 & \text{otherwise.}\end{cases}$$

Initially, we set $e(k) = 0$ if k is not in a run. In particular, note that if r is at the end of the run $R(r,s)$ then $e(r) = 0$. The use of the express vector results in a particular type of path compression in the m-tree, which maintains the structure that is crucial in the solution of the intersection problem.

Suppose a run $R(r,s) \cap \mathcal{C}_k \neq \emptyset$. We then (temporarily) set $e(r) = -\min\{\ell \in R(r,s) \cap \mathcal{C}_k\}$; that is, $-e(r)$ points to the lowest numbered vertex in the run that is also in $\mathcal{C}_k$. This can be determined easily as the marker array c is being computed (for each $i \in \mathcal{C}_k$, check if $e(i) > 0$). Given the arrays e, c, and m, and the integer $|\mathcal{D}_{ij}|$, the following procedure computes q_{ijk}.

Procedure Get_q_{ijk}.

```
(G1)    $q \leftarrow j$; $count \leftarrow 1$
(G2)    while $c(q) = 0$ and $count \le |\mathcal{D}_{ij}|$ do
(G3)        if $e(q) \le 0$ then
(G4)            $q \leftarrow m(q)$
(G5)            $count \leftarrow count + 1$
(G6)        else
(G7)            $q' \leftarrow q$
(G8)            $q \leftarrow e(q)$
(G9)            if $e(q) < 0$ then $q \leftarrow -e(q)$
(G10)           $count \leftarrow count + q - q'$
(G11)       end if
(G12)   end while
(G13)   if $c(q) = 0$ or $count > |\mathcal{D}_{ij}|$ , then $q \leftarrow 0$
```

This procedure simply generates the sequence (7) for $\mathcal{D}_{ij}$, looking for q_{ijk}. When it is possible to do so, we use the express vector to process runs in $O(1)$ work.

5. The intersection problem for grid graphs. Let $\mathcal{G}(k)$ be the $n \times n$ grid graph with $n = 2^k - 1$, ordered using nested dissection ($N = n^2$). $\mathcal{G}'(k)$ and $\mathcal{T}(k)$ will denote the triangulation of $\mathcal{G}(k)$ and its m-tree, respectively. Recall that nested dissection orders the vertices in a cross-shaped separator $\mathcal{S}$, consisting of the $2n-1$ vertices in row $(n+1)/2$ and column $(n+1)/2$ last. This leaves four grid graphs $\mathcal{G}'(k-1)$ to be recursively ordered. This is shown schematically in Fig. 6.

The recursive nature of the ordering is reflected in the m-tree which has the recursively defined structure shown in Fig. 7.

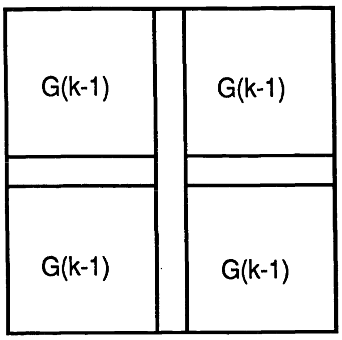


FIG. 6. *Nested dissection ordering.*

The nodes labeled $\mathcal{T}(k-1)$ are m-trees for the four subgraphs $\mathcal{G}'(k-1)$. Note that, with the exceptions of vertices x_{p+m} and x_n, $m(j) = j+1$ for all $x_j \in \mathcal{S}$.

LEMMA 5.1. *For each $x_i \in \mathcal{X}$ in $\mathcal{G}(k)$, $|\mathcal{L}_i| \leq 2$.*

Proof. Since $|\mathcal{L}'_i| \leq 4$, there are at most four possible leaves for any vertex. The proof is by induction on k; the case $k = 1$ is trivial. Using the induction hypothesis for the four subgraphs $\mathcal{G}(k-1)$, we are left to consider only $x_i \in \mathcal{S}$. For such an x_i, at most two vertices in $\mathcal{L}'_i$ are not also in $\mathcal{S}$. By considering all the special cases, we straightforwardly surmise that $|\mathcal{L}_i| \leq 2$ for all $x_i \in \mathcal{S}$. □

LEMMA 5.2. *The height $h(k)$ of $\mathcal{T}(k)$ is*

$$h(k) = 3(2^k - 2) - 2(k-1) \tag{12}$$

Proof. Evidently

$$h(k) = h(k-1) + 3 * 2^{k-1} - 2.$$

with $h(1) = 0$. The solution of the difference equation is given in (12). □

Note also from (12) that

$$h(k) \leq 3n.$$

Thus we have the following lemma from Lemmas 5.1 and 5.2.

LEMMA 5.3. *For any $x_i \in \mathcal{G}'(k)$,*

$$|\mathcal{C}_i| \leq 6n.$$

THEOREM 5.4. *The complexity of the intersection problem without using the express function is $O(n^3)$.*

Proof. We estimate the cost $F(n)$ for computing q_{ijk} for all the relevant indices within the context of the numerical factorization. The procedure uses only the m-function, and thus must generate all entries in (8).

First, consider the cost for a single $x_i \in \mathcal{S}$. By Lemmas 5.1–5.3, the cost will be at most $O(n^2)$, since $|\mathcal{L}_j|$ and $|\mathcal{L}'_j|$ are bounded by constants, and $|\mathcal{C}_j| \leq O(n)$ for all $x_j \in \mathcal{X}$. Since $|\mathcal{S}| = 2n - 1$, the cost for all vertices in $\mathcal{S}$ is $O(n^3)$. Thus,

$$F(n) \leq 4F(n/2) + \gamma n^3 \tag{13}$$

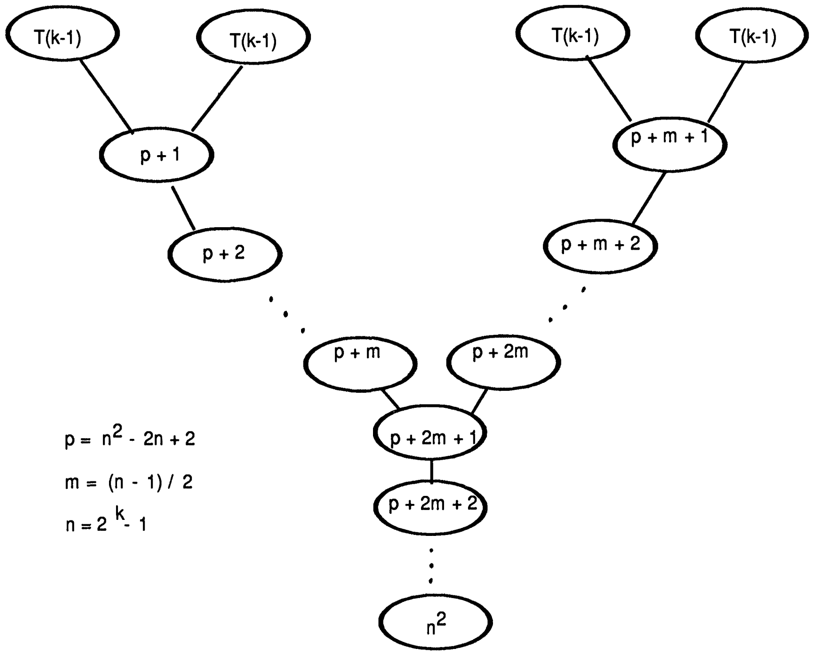


FIG. 7. *m-tree for the nested dissection ordering.*

for some constant γ. The solution of the majorizing difference equation shows $F(n) \leq O(n^3)$. This is the same complexity as the numerical factorization. Thus, although the intersection problem contributes to the highest order complexity term, it does not increase the overall order of complexity. □

We now consider solving the intersection problem using the express vector and the m-function, as in Procedure Get_q_{ijk}. We let $Q(k)$ be the cost of Procedure Get_q_{ijk} for $\mathcal{T}(k)$. Then, for any j, the cost of processing the portion of the first sequence in (8) that lies in $\mathcal{S}$ is $O(1)$, since all but two vertices in $\mathcal{S}$ are in runs. Thus

$$Q(k) \leq Q(k-1) + \gamma$$

for some constant γ. The solution of this difference equation shows

$$Q(k) \leq \gamma k + O(1) = O(\log n).$$

Using Lemmas 5.1-5.3, we have

$$F(n) \leq 4F(n/2) + \gamma n^2 \log n$$

instead of (13). The solution of the majorizing difference equation shows

$$F(n) \leq O(n^2 (\log n)^2).$$

Thus we have shown the following theorem.

THEOREM 5.5. *The complexity of the intersection problem using the express function is* $O(n^2(\log n)^2)$.

In this case the intersection problem does not contribute to the highest order complexity terms.

TABLE 1
Nonnumerical overhead for grid graphs.

k	3	4	5	6	7	8
N	49	225	961	3,969	16,129	65,025
f	580	11,496	153,668	1,664,596	15,963,924	142,335,428
m	448	6,684	66,392	524,564	3,585,936	22,215,844
e	72	2,256	35,460	332,716	2,519,844	16,693,084
s	66	3,814	95,094	1,510,526	18,581,430	195,752,462

In Table 1, we compare the actual nonnumerical overhead for the two procedures for $n \times n$ grid graphs with $n = 2^k - 1$, $3 \leq k \leq 8$, using the nested dissection ordering. The row labeled f lists the number of floating point operations used on line (S4) of Procedure Sparse Factor; this also counts the number of unavoidable indirect addresses corresponding to indices in the intersection $\mathcal{C}_i \cap \mathcal{C}_j$. The row labeled m counts the number of times line (G4) of Procedure Get_q_{ijk} is executed, while the row labeled e counts the number of times line (G8) is executed. Thus the sum $m + e$ reflects the nonnumerical overhead associated with solving the intersection problem using the express vector. Finally, the row labeled s gives the number of saved indirect addresses; $m + e + s$ reflects the nonnumerical overhead in solving the intersection problem without the express vector. This can be modeled using Procedure Get_q_{ijk} with $e(i) = 0$ for all i.

In Table 1, both f and s grow as $O(n^3)$ complexity; $m+e$ is growing as $O(n^2(\log n)^2)$. The behavior illustrated here seems to be typical of general grid problems arising from finite element or finite difference discretizations of partial differential equations. We have had the most experience with problems posed on irregular and nonuniform triangular meshes; there we have noticed that without the express vector, in large problems, $50 - 60$ percent of the indirect addresses (generated as $k \leftarrow m(k)$) used in solving the problem do not have corresponding floating point operations. On the other hand, with the express vector, the ratio of indirect addresses to floating point operations is close to one (or one half for nonsymmetric problems retaining a symmetric zero-nonzero structure). Thus, a general sparse matrix code based on the bordering algorithm should have execution times comparable to the current generation of general sparse matrix packages based on rowwise Cholesky factorization (e.g., Yale Sparse Matrix Package [4] and Sparspak [5]).

6. A note on data structures and implementation. Although it is possible to implement the sparse bordering algorithm using only $O(N)$ integer storage for the factored matrix (all temporary work space), we favor the data structure described in [3], which requires an integer array of length $NZ + 1$, where NZ is the number of nonzeros in the upper triangle of A. This allows for a relatively simple and more time-efficient code. At the same time, $NZ = O(N)$ for many sparse matrix problems, for example, systems arising from discretizations of partial differential equations. We briefly summarize the data structures proposed in [3], restricted here to the case of symmetric matrices, and discuss the practical implementation of our procedures within this framework. [1]

Nonzeros in the upper triangle of the sparse matrix A are stored in an array a

[1] A prototype package for carrying out general sparse Gaussian elimination using the bordering approach (essentially identical to that presented in [3] except for the inclusion of an express vector in the numerical factorization routine) is available from Argonne National Laboratory via Netlib.

of length $NZ + 1$; elements are referenced through an integer array ja, also of length $NZ+1$. Nonzeros in the Cholesky factor $L = U^t$ and the diagonal matrix D are stored in an array u of length $NZ' + 1$, where NZ' is the number of nonzeros in $D + U$. Entries in u are referenced through ja and an integer array jl of length $NZ + 1$, the same length as ja.

The entries of a and ja are defined as follows: $a(i), 1 \leq i \leq N$ contain the ith diagonal entry of A (a_{ii}). Entries $a(ja(i))$ to $a(ja(i+1) - 1)$, $1 \leq i \leq N$ contain the nonzeros in the strict upper triangular part of the ith column of A, stored in order of decreasing row index; corresponding entries of the ja array contain the row indices. The entry $a(N + 1)$ is arbitrary and is included because $N + 1$ pointers are required at the beginning of ja. This scheme, although different in detail, was motivated by the data structures of the Yale Sparse Matrix Package [4], except that the roles of rows and columns have been interchanged.

The array u is somewhat similar in structure to a. The first N locations of u contain the reciprocals of the diagonal entries of D; $u(N + 1)$ is arbitrary. The following entries in u contain the nonzeros in the strict upper triangular part of U, stored column by column; the entries are ordered corresponding to the sorting of the indices generated by our solution of the sorting problem. The diagonal entries of U are unity and are not stored.

The array jl is used in conjunction with ja to access the data in u. Recall that each row index stored in ja corresponds to one of the generalized leaves in the sets $\mathcal{L}'_i$; this is really the key observation in reducing the overhead storage. Entries $jl(i), 1 \leq i \leq N-1$ contain $m(i)$, yielding the $N-1$ edges in the m-tree. Entry $jl(N)$ is arbitrary. Entries $jl(i-1)$ (and $jl(i) - 1$), $N+2 \leq i \leq NZ+1$ point at the first (and last) entries in u where the nonzeros of U generated by the leaf index $ja(i)$ are stored. These nonzeros correspond to one of the ordered index sets $\mathcal{D}_{jk}$ defined in this work. The row index for the first entry is of course given by $ja(i)$; subsequent indices are generated in the proper order using the m-tree.

The complete set of data structures as they appear for our simple 3×3 grid graph is shown in Table 2.

The following looping structure accesses the nonzeros in column k of U in sorted order:

```
(A1)    for $leaf = ja(k+1) - 1, ja(k)$, step $-1$
(A2)        $j \leftarrow ja(leaf)$
(A3)        for $jloc = jl(leaf - 1), jl(leaf) - 1$, step 1
(A4)            ($u(jloc)$ contains matrix entry $u_{jk}$)
(A5)            $j \leftarrow jl(j)$
(A6)        end for
(A7)    end for
```

Statement (A1) loops over the generalized leaves for $\mathcal{T}_k$. The loop (A3)-(A6) generates the ordered index set $\mathcal{D}_{kj}$ associated with that leaf; the m-tree is used in (A5). By the definition of jl, the parameter $jloc$ always points to the entry in the Cholesky factor $u_{jk} = \ell_{kj}$.

Variations on the looping structure (A1)-(A7) suffice for the forward and backward solution procedures using this data structure; for these procedures, the leaves can be processed in either forward or reverse order. This also suffices for the middle loop, line (S3) in Procedure Sparse Factor. The inner loop on line (S4) is of this form,

TABLE 2
Data structure entries for 3 × 3 *grid graph.*

i	$ja(i)$	$a(i)$	$jl(i)$	$u(i)$
1	11	a_{11}	5	d_{11}^{-1}
2	11	a_{22}	5	d_{22}^{-1}
3	11	a_{33}	6	d_{33}^{-1}
4	11	a_{44}	6	d_{44}^{-1}
5	11	a_{55}	7	d_{55}^{-1}
6	13	a_{66}	7	d_{66}^{-1}
7	15	a_{77}	8	d_{77}^{-1}
8	17	a_{88}	9	d_{88}^{-1}
9	20	a_{99}		d_{99}^{-1}
10	23		11	
11	2	a_{25}	12	u_{25}
12	1	a_{15}	13	u_{15}
13	4	a_{46}	14	u_{46}
14	3	a_{36}	15	u_{36}
15	4	a_{47}	17	u_{47}
16	1	a_{17}	19	u_{67}
17	7	a_{78}	20	u_{17}
18	6	a_{68}	21	u_{57}
19	5	a_{58}	22	u_{78}
20	8	a_{89}	23	u_{68}
21	3	a_{39}	26	u_{58}
22	2	a_{29}	28	u_{89}
23				u_{39}
24				u_{69}
25				u_{79}
26				u_{29}
27				u_{59}

except that the starting value for the line corresponding to (A3) (i.e., $jl(leaf-1)$) is replaced by a value determined by Procedure Get_q_{ijk}. For this data structure, a simple modification of parameter *count* in Procedure Get_q_{ijk} will allow *count* to return the starting index for line (A3), a value $jl(leaf-1) \le count \le jl(leaf)-1$, provided that the intersection is not empty.

REFERENCES

[1] R. E. BANK, PLTMG *users' guide, edition 5.0*, Tech. report, Department of Mathematics, University of California, San Diego, CA, 1988.

[2] R. E. BANK, T. F. DUPONT, AND H. YSERENTANT, *The hierarchical basis multigrid method*, Numer. Math., 52 (1988), pp. 427–458.

[3] R. E. BANK AND R. K. SMITH, *General sparse elimination requires no permanent integer storage*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 574–584.

[4] S. C. EISENSTAT, M. C. GURSKY, M. SCHULTZ, AND A. SHERMAN, *Algorithms and data structures for sparse symmetric Gaussian elimination*, SIAM J. Sci. Statist. Comput., 2 (1982), pp. 225–237.

[5] A. GEORGE AND J. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Pren-

tice Hall, Englewood Cliffs, NJ, 1981.

[6] K. H. LAW AND S. J. FENVES, *A node addition model for symbolic factorization*, ACM TOMS, 12 (1986), pp. 37–50.

[7] J. W. H. LIU, *A compact row storage scheme for Cholesky factors using elimination trees*, ACM TOMS, 12 (1986), pp. 127–148.

[8] ———, *The role of elimination trees in sparse factorization*, Tech. Report CS-87-12, Department of Computer Science, York University, Ontario, Canada, 1987.

[9] D. J. ROSE, *A graph theoretic study of the numeric solution of sparse positive definite systems*, in Graph Theory and Computing, Academic Press, New York, 1972.

[10] D. J. ROSE, R. E. TARJAN, AND G. S. LUEKER, *Algorithmic aspects of vertex elimination on graphs*, SIAM J. Comput., 5 (1976), pp. 226–283.

[11] D. J. ROSE AND G. F. WHITTEN, *A recursive analysis of disection strategies*, in Sparse Matrix Computations, Academic Press, New York, 1976.

[12] R. SCHRIEBER, *A new implementation of sparse Gaussian elimination*, ACM TOMS, 8 (1982), pp. 256–276.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 1, pp. 161-173, January 1990

AN $O(N^2)$ METHOD FOR COMPUTING THE EIGENSYSTEM OF $N\times N$ SYMMETRIC TRIDIAGONAL MATRICES BY THE DIVIDE AND CONQUER APPROACH*

DORON GILL† AND EITAN TADMOR‡

To Eugene Isaacson on his 70th birthday

Abstract. An efficient method to solve the eigenproblem of $N \times N$ symmetric tridiagonal matrices is proposed. Unlike the standard eigensolvers that necessitate $O(N^3)$ operations to compute the eigenvectors of such matrices, the proposed method computes both the eigenvalues and eigenvectors with only $O(N^2)$ operations. The method is based on *serial* implementation of the recently introduced Divide and Conquer algorithm [3], [1], [4]. It exploits the fact that by $O(N^2)$ Divide and Conquer operations one can compute the eigenvalues of an $N \times N$ symmetric tridiagonal matrix *and* a small number of pairs of successive rows of its eigenvector matrix. The rest of the eigenvectors (either all together or one at a time) are computed by linear three-term recurrence relations. The paper is concluded with numerical examples that demonstrate the superiority of the proposed method for a special class of symmetric tridiagonal matrices, by saving an *order* of magnitude in execution time at the expense of sacrificing a *few* orders of accuracy, although for symmetric tridiagonal matrices in general, the method appears to be unstable.

Key words. symmetric eigenvalue problem, divide and conquer, updating problem

AMS(MOS) subject classification. 65F15

1. Introduction. The QR algorithm computes the eigenvalues of an $N \times N$ Symmetric Tridiagonal (ST) matrix with $O(N^2)$ operations, while the corresponding eigenvector matrix is accumulated during the algorithm at the expense of $O(N^3)$ operations. The additional order of magnitude required to compute the eigenvectors is typical of serial algorithms. A complete $O(N^2)$ eigensolver can be obtained by appending such serial algorithms with the Inverse Iteration (INVIT) method. Indeed, $O(N)$ operations of only *one* INVIT will suffice to accurately compute each eigenvector corresponding to an *isolated* eigenvalue [8, Chap. 4]. In case of clustered eigenvalues, however, the INVIT requires a more carefully chosen initialization, to avoid the loss of mutual orthogonality between the corresponding, closely "related" eigenvectors.

Recently, a parallel Divide and Conquer (DC) algorithm was introduced for computing the spectral decomposition of ST matrices [3], [1], [4]. A *serial* implementation of this algorithm, described in § 2, requires the same number of operations. Namely, the eigenvalues, which coincide with the roots of the so-called secular equation [6], are computed at the expense of no more than $O(N^2)$ sequential operations, while the associated eigenvectors necessitate $O(N^3)$ sequential operations. As before, the INVIT, taken with the necessary precautions, is available here as an $O(N^2)$ method to compute these eigenvectors. In §§ 3-4, we propose an alternative efficient method, derived from

* Received by the editors March 2, 1988; accepted for publication (in revised form) January 31, 1989.

† School of Mathematical Sciences, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel Aviv University, Ramat Aviv, Tel Aviv 69 978, Israel.

‡ School of Mathematical Sciences, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel Aviv University, Ramat Aviv, Tel Aviv 69 978, Israel. Present address, Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, Virginia 23665. The research of this author was partially supported by the National Aeronautics and Space Administration under NASA contract NAS1-18107. Additional support was provided by the U.S.-Israel Binational Science Foundation grant 85-00346, and by National Science Foundation grant DMS85-03294, while this author was in residence at the University of California at Los Angeles, Los Angeles, California 90024.

(and therefore better suited to) the DC algorithm, which computes the eigensystem of $N \times N$ ST matrices with only $O(N^2)$ sequential operations. The method employs linear three-term recurrence relations that successively compute the *rows* of the eigenvector matrix (or the *components* of each of the desired eigenvectors). The coefficients of these relations depend on the already computed eigenvalues, and the method hinges on the fact that the initial first two rows (or components) for the recurrence relations *emerge naturally* from the DC computation of these eigenvalues. Thus, the input data for the recurrence relations depends solely on the $O(N^2)$ operations for the DC calculation of the eigenvalues. Together with the additional $O(N^2)$ operations required to carry out these relations, we end up with an efficient $O(N^2)$ method to compute the whole eigensystem of *ST* matrices. It should be emphasized that the advantages of the DC algorithm are retained in our case. That is, we have a method which on the one hand is well suited to exploit parallelism; on the other hând, even when run in serial mode on large problems, the method is faster than the previously best sequential algorithms, e.g., [3], [4].

The main limitation of the proposed method lies in the possible instability of the three term recurrence relations mentioned above. In § 4, we identify a useful class of ST matrices for which the corresponding recurrence relations are stable. In such stable cases, the numerical results of our method are almost as accurate as the standard DC algorithm. In the general case, however, the accuracy of our method may deteriorate for large N, $N \gtrsim 100$, due to the instability of the corresponding recurrence relations. To overcome the unstable error accumulation in such cases, one may *restart* the recurrence relations at any stage of the recursive iterations with two new successive rows of the eigenvector matrix. In § 4, we show how to obtain two such successive rows for restarting, at the expense of $O(N^2)$ DC operations.

Due to the sensitivity of the three-term recurrence relations, their input data should be provided with high accuracy. To achieve this, we employ in § 5 an improved root finder—interesting for its own sake—in order to solve the secular equation mentioned above. Numerical examples that demonstrate the efficiency as well as the limitations of the proposed method are presented in § 6.

2. The Divide and Conquer algorithm—An overview. Let D_N be an $N \times N$ diagonal matrix and let $D_N + \sigma z_N z_N^t$ be a Rank One Modification (ROM) of this matrix by a unit N-vector z_N.[1] The spectral decomposition of such ROM matrices is the heart of the Divide and Conquer (DC) algorithm. Here we note that the problem of finding the spectral decomposition of an N-dimensional ROM matrix, the so-called *updating problem*, can be solved at the expense of no more than Const. N^2 operations [1], [3], [4]. Details of this solution are discussed in § 4.

With this in mind we now turn to consider the eigenproblem of general $N \times N$ Symmetric Tridiagonal (ST) matrices

$$T_N = \left[\begin{array}{ccc|ccc} t_{11} & t_{12} & & & & \\ t_{21} & t_{22} & & & & \\ & & t_{mm} & t_{mm+1} & & \\ \hline & & t_{m+1,m} & t_{m+1,m+1} & & \\ & & & & & t_{N-1,N} \\ & & & & t_{N,N-1} & t_{NN} \end{array}\right], \qquad t_{ij} = t_{ji}.$$

[1] Throughout the paper, vectors and matrices will be used with a subscript index denoting their dimension.

We can assume without restriction that N is even, $N=2m$, and that T_N is already given in its unreduced form, i.e., $t_{i,i+1}\neq 0$, $1\leqq i\leqq N-1$; otherwise, T_N is decoupled into smaller unreduced ST matrices. Then, we can *split* T_N into the sum of

$$T_N=\left[\begin{array}{ccc:ccc} t_{11} & t_{12} & & & & \\ t_{22} & & & & & \\ & & t_{mm}-\beta & & 0 & \\ \hdashline & & 0 & t_{m+1,m+1}-\beta & & \\ & & & & & \\ & & & & & t_{N,N} \end{array}\right]+\beta\left[\begin{array}{ccc} 0 & \vdots & 0 \\ & 1\ 1 & \\ \hdashline & 1\ 1 & \\ 0 & \vdots & 0 \end{array}\right],\qquad \beta=t_{m,m+1},$$

i.e.,

$$(2.1)\qquad T_N=\begin{bmatrix} T^{(1)}_{N/2} & \\ & T^{(2)}_{N/2}\end{bmatrix}+\beta b_N b_N^t,\qquad b_N=e_N^{(m)}+e_N^{(m+1)},$$

where the blocks $T^{(1)}_{N/2}$ and $T^{(2)}_{N/2}$ are $N/2\times N/2$ ST matrices and $\beta\equiv t_{m,m+1}\neq 0$ is the coupling term of these two blocks.

The DC algorithm [3], [1], [4] is based on the fact that in order to solve the eigenproblem of N-dimensional ST matrices, it is sufficient to solve this problem for $(N/2)$-dimensional ST matrices. Specifically, if

$$(2.2)\qquad \begin{aligned} T^{(1)}_{N/2}&=P^{(1)}_{N/2}\Lambda^{(1)}_{N/2}P^{(1)\prime}_{N/2}, & P^{(1)}_{N/2}P^{(1)\prime}_{N/2}&=I_{N/2}\\ T^{(2)}_{N/2}&=P^{(2)}_{N/2}\Lambda^{(2)}_{N/2}P^{(2)\prime}_{N/2}, & P^{(2)}_{N/2}P^{(2)\prime}_{N/2}&=I_{N/2},\end{aligned}$$

are the spectral decompositions of the $N/2\times N/2$ ST matrices $T^{(1)}_{N/2}$ and $T^{(2)}_{N/2}$, respectively, then we can compute the spectral decomposition of the $N\times N$ ST matrix T_N by the following procedure:

I. First, we evaluate the unit N-vector z_N,

$$(2.3a)\qquad z_N=\frac{1}{\sqrt{2}}\begin{bmatrix} P^{(1)}_{N/2} & \\ & P^{(2)}_{N/2}\end{bmatrix}^t b_N,\qquad b_N=\begin{bmatrix}0\\ \cdot\\ 1\\ \cdots\\ 1\\ \cdot\\ 0\end{bmatrix}$$

so that by (2.1), (2.2), and (2.3a), T_N is unitarily similar to the ROM matrix

$$\begin{aligned} T_N&=\begin{bmatrix} P^{(1)}_{N/2}\Lambda^{(1)}_{N/2}P^{(1)t}_{N/2} & \\ & P^{(2)}_{N/2}\Lambda^{(2)}_{N/2}P^{(2)t}_{N/2}\end{bmatrix}+\beta b_N b_N^t\\ &=\begin{bmatrix} P^{(1)}_{N/2} & \\ & P^{(2)}_{N/2}\end{bmatrix}\left(\begin{bmatrix} \Lambda^{(1)}_{N/2} & \\ & \Lambda^{(2)}_{N/2}\end{bmatrix}+2\beta z_N z_N^t\right)\begin{bmatrix} P^{(1)t}_{N/2} & \\ & P^{(2)t}_{N/2}\end{bmatrix}\\ &=\begin{bmatrix} P^{(1)}_{N/2} & \\ & P^{(2)}_{N/2}\end{bmatrix}(D_N+2\beta z_N z_N^t)\begin{bmatrix} P^{(1)}_{N/2} & \\ & P^{(2)}_{N/2}\end{bmatrix}^t,\\ D_N&=\begin{bmatrix} \Lambda^{(1)}_{N/2} & \\ & \Lambda^{(2)}_{N/2}\end{bmatrix}.\end{aligned}$$

II. Second, we solve the updating problem by finding the spectral decomposition of the ROM matrix

$$(2.3b)\qquad D_N+\sigma z_N z_N^t=Q_N\Lambda_N Q_N^t,\qquad Q_N Q_N^t=I_N,\quad \sigma=2\beta.$$

III. Finally, we compute the unitary matrix

$$P_N = \begin{bmatrix} P^{(1)}_{N/2} & \\ & P^{(2)}_{N/2} \end{bmatrix} Q_N \tag{2.3c}$$

and obtain, by (2.3b) and (2.3c), the spectral decomposition of T_N as

$$T_N = \begin{bmatrix} P^{(1)}_{N/2} & \\ & P^{(2)}_{N/2} \end{bmatrix} Q_N \Lambda_N Q^t_N \begin{bmatrix} P^{(1)}_{N/2} & \\ & P^{(2)}_{N/2} \end{bmatrix}^t = P_N \Lambda_N P^t_N, \qquad P_N P^t_N = I_N.$$

This process can be applied recursively: the N-dimensional eigenproblem of T_N is solved in terms of two independent $(N/2)$-dimensional eigenproblems of $T^{(1)}_{N/2}$ and $T^{(2)}_{N/2}$, which in turn are solved in terms of four independent $(N/4)$-dimensional eigenproblems of $T^{(1)}_{N/4}$, $T^{(2)}_{N/4}$, $T^{(3)}_{N/4}$, $T^{(4)}_{N/4}$, etc. Thus, the DC algorithm for an $N=2^n$-dimensional ST matrix T_N is organized as follows. After $n-1$ splitting steps we are left with 2^{n-2} pairs of 2×2 ST matrices. In the first iteration they are used to construct, with the help of (2.3a)-(2.3c), the eigensystem of 2^{n-3} pairs of 4×4 ST matrices; in the second iteration, one constructs the eigensystem of 2^{n-4} pairs of 8×8 ST matrices, etc.; after $n-2$ such iterations we end up with the eigensystems of the pair $T^{(1)}_{N/2}$, $T^{(2)}_{N/2}$, and the last $n-1$ iteration solves the eigenproblem of T_N. A sequential implementation of a typical kth iteration consists of 2^{n-k-1} times, evaluating the 2^{k+1}-dimensional unit vectors z in the first stage (2.3a), solving 2^{k+1}-dimensional ROM eigenproblems in the second stage (2.3b), and computing 2^{k+1}-dimensional products of unitary matrices in the third stage (2.3c).

The total amount of work spent on the first two stages, (2.3a) and (2.3b), of all iterations, does not exceed $2\,\text{Const.}\,N^2$; the total work required for computing the eigenvectors in (2.3c) is $\sum_{k=1}^{n-1} 2^{n-k-1}\cdot 4(2^k)^3 \lesssim \frac{2}{3}N^3$. Thus, the total operations cost of the DC algorithm for finding the eigensystem (both the eigenvalues and eigenvectors) of an $N\times N$ ST matrix is $\frac{2}{3}N^3 + 2\,\text{Const.}\,N^2$.

If only the eigenvalues are required, then we can do better by saving the $O(N^3)$ operations required to compute the eigenvectors in the third stage (2.3c). Instead, the first stage of a typical kth iteration, which requires 2^{n-k-1} different evaluations of 2^{k+1}-dimensional unit vectors of the form

$$z_{2^{k+1}} = \frac{1}{\sqrt{2}} \begin{bmatrix} P^{(1)}_{2^k} & \\ & P^{(2)}_{2^k} \end{bmatrix}^t b_{2^{k+1}},$$

can be efficiently implemented as follows: According to (2.3c), $P^{(1)}_{2^k}$ is represented by a successive product of

$$\begin{bmatrix} Q^{(1)}_{2^j} & & \\ & \ddots & \\ & & Q^{(2^{k-j})}_{2^j} \end{bmatrix} \qquad j=k, k-1, \cdots, 1,$$

where $Q^{(\cdot)}_{2^j}$ were found by spectral decompositions of ROM matrices in previous iterations; similarly, $P^{(2)}_{2^k}$ is represented by a successive product of

$$\begin{bmatrix} Q^{(2^{k-j}+1)}_{2^j} & & \\ & \ddots & \\ & & Q^{(2^{k-j+1})}_{2^j} \end{bmatrix}, \qquad j=k, k-1, \cdots, 1.$$

Hence, we can evaluate each of the 2^{n-k-1} different vectors, $z_{2^{k+1}}$, as $z_{2^{k+1}} = z^{(k)}_{2^{k+1}}$, where

$$z^{(j)}_{2^{k+1}} = \begin{bmatrix} Q^{(1)}_{2^j} & & \\ & \ddots & \\ & & Q^{(2^{k-j+1})}_{2^j} \end{bmatrix}^t z^{(j-1)}_{2^{k+1}}, \qquad j=1,2,\cdots,k, \qquad z^{(0)}_{2^{k+1}} \equiv \frac{1}{\sqrt{2}} b_{2^{k+1}}, \tag{2.4a}$$

at the expense of $\sum_{j=1}^{k} 2^{k-j+1} \cdot 2^{2j} \lesssim 4^{k+1}$ operations. The total work spent on the first stages in all iterations is therefore $\sum_{k=1}^{n-1} 2^{n-k-1} \cdot 4^{k+1} \lesssim 2N^2$. This is complemented with the solution of 2^{n-k-1} different updating problems (see (2.3b))

$$D_{2^{k+1}} + \sigma z_{2^{k+1}} z_{2^{k+1}}^t = Q_{2^{k+1}} \Lambda_{2^{k+1}} Q_{2^{k+1}}. \tag{2.4b}$$

The total work spent on the second stage in all iterations amounts to 2 Const. N^2. Consequently, the total operations cost of the DC algorithm, (2.4a), (2.4b), for finding the eigenvalues of an $N \times N$ ST matrix is (2 Const.+2)N^2.

3. An $O(N^2)$ method for the eigensystem of $N \times N$ ST matrix. Given an $N \times N$ ST matrix T_N, we can compute its eigenvalues by the DC algorithm (2.4a), (2.4b) at the expense of no more than $O(N^2)$ operations.[2] Thus, it remains to compute efficiently, i.e., with $O(N^2)$ operations, the eigenvectors of this matrix. To this end we may proceed as follows.

We seek the unitary matrix P_N, $P_N P_N^t = I_N$, which diagonalizes T_N,

$$T_N = P_N \Lambda_N P_N^t. \tag{3.1}$$

Let $p^{(i)} \equiv p_N^{(i)}$ denote the ith *row* vector of P_N. Equating the ith rows of

$$T_N P_N = P_N \Lambda_N$$

we obtain, in view of the reduced tridiagonal structure of T_N,

$$t_{i,i-1} p_N^{(i-1)} + t_{i,i} p_N^{(i)} + t_{i,i+1} p_N^{(i+1)} = p_N^{(i)} \Lambda_N, \qquad t_{i,i\pm 1} \neq 0. \tag{3.2}$$

Equation (3.2) is a linear three-term recurrence relation between the rows, $p_N^{(i)}$, of P_N, whose coefficients are determined by the entries of T_N. The input data required to solve these relations uniquely consists of

(1) The eigenvalues $\Lambda_N = \text{diag}(\lambda^{(1)}, \lambda^{(2)}, \cdots, \lambda^{(N)})$ of T_N, which determine the terms $p_N^{(i)} \Lambda_N \equiv (\lambda^{(1)} p_{i1}, \lambda^{(2)} p_{i2}, \cdots, \lambda^{(N)} p_{iN})$ on the right of (3.2). The eigenvalues are computed by the DC algorithm (2.4a), (2.4b) with (2 Const.+2)N^2 operations.

(2) Two successive *rows* of P_N that will serve as initial data for the recursive three-term relations (3.2). The proposed method hinges on the observation that two such rows *emerge naturally* from that part of the DC algorithm (2.4a), (2.4b) which computes the eigenvalues of T_N. Indeed, from the last $n-1$ iteration of (2.4a) we have at our disposal the unit N-vector z_N, which according to (2.3a) satisfies

$$\begin{bmatrix} z_{N/2}^{(1)} \\ z_{N/2}^{(2)} \end{bmatrix} = \begin{bmatrix} P_{N/2}^{(1)} & \vdots & \\ \cdots & \cdots & \cdots \\ & \vdots & P_{N/2}^{(2)} \end{bmatrix}^t \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \qquad \begin{bmatrix} z_{N/2}^{(1)} \\ z_{N/2}^{(2)} \end{bmatrix} \equiv \sqrt{2} \cdot z_N. \tag{3.3}$$

Hence $z_{N/2}^{(1)}$ and $z_{N/2}^{(2)}$ are in fact the last and first column vectors of $P_{N/2}^{(1)t}$ and $P_{N/2}^{(2)t}$, respectively. Put differently, $(z_{N/2}^{(1)}, O_{N/2})^t$ and $(O_{N/2}, z_{N/2}^{(2)})^t$ are row numbers $m = N/2$ and $m+1$ of

$$\begin{bmatrix} P_{N/2}^{(1)} & \\ & P_{N/2}^{(2)} \end{bmatrix}.$$

[2] In fact, as observed by Cuppen [3], this number of operations can be substantially reduced by up to $O(N \log N)$ operations, in practical cases which employ sufficiently many deflations.

Consequently, equating the m and $m+1$ rows of (2.3c), we obtain the two initial successive rows as

$$p_N^{(m)}=(z_{N/2}^{(1)}, O_{N/2})^t Q_N,\qquad p_N^{(m+1)}=(O_{N/2}, z_{N/2}^{(2)})^t Q_N; \tag{3.4}$$

the remaining rows of P_N are computed recursively by (3.2)

$$p_N^{(i+1)}=\frac{1}{t_{i,i+1}}[p_N^{(i)}(\Lambda_N-t_{i,i}I_N)-t_{i,i-1}p_N^{(i-1)}],\qquad i=m+1,\cdots,N-1, \tag{3.5a}$$

$$p_N^{(i-1)}=\frac{1}{t_{i,i-1}}[p_N^{(i)}(\Lambda_N-t_iI_N)-t_{i,i+1}p_N^{(i+1)}],\qquad i=m,m-1,\cdots,2. \tag{3.5b}$$

The operation cost of (3.4)-(3.5) does not exceed $3N^2$. Thus (2.4a), (2.4b) together with (3.4), (3.5) provide us with an $O(N^2)$ method for computing the whole eigensystem of $N\times N$ ST matrices.

The error analysis of the proposed method depends on two ingredients:

(1) The accuracy of the input data for (3.5a), (3.5b), namely, the errors accumulated in computing the eigenvalues $\Lambda_N=\text{diag}\,(\lambda^{(1)},\lambda^{(2)},\cdots,\lambda^{(N)})$ and the two successive rows $p_N^{(m)}$, $p_N^{(m+1)}$ of P_N. The size of these errors is determined by the stability properties of the DC algorithm (2.4a), (2.4b). In this context, we recall that stable behavior of the DC algorithm hinges on an accurate solution of the ROM problem (2.4b) (see [1], [3], [4]). In § 5, we borrow from [1], [3], and [4], discussing a root finder for an accurate computation of the eigenvalues $\Lambda_{2^{k+1}}$, which are obtained as the roots of the characteristic equation associated with the ROM matrix in (2.4b).

(2) The second source of error is due to accumulation of rounding errors in the recurrence relations (3.5a), (3.5b). In order to examine this error accumulation, we rewrite (3.5) as a one-step iteration

$$[p_N^{(i+1)},p_N^{(i)}]=[p_N^{(i)},p_N^{(i-1)}]\begin{bmatrix}1/t_{i,i+1}[\Lambda_N-t_{i,i}I_N] & I_N\\ -(t_{i,i-1}/t_{i,i+1})I_N & O_N\end{bmatrix},\quad i=m+1,\cdots,N-1, \tag{3.6a}$$

$$[p_N^{(i-1)},p_N^{(i)}]=[p_N^{(i)},p_N^{(i+1)}]\begin{bmatrix}1/t_{i,i-1}[\Lambda_N-t_{i,i}I_N] & I_N\\ -(t_{i,i+1}/t_{i,i-1})I_N & O_N\end{bmatrix},\quad i=m,m-1,\cdots,2. \tag{3.6b}$$

An indication of the stability properties of (3.6a), (3.6b) is provided by the eigenvalues $\kappa=\kappa_{ij}^{\pm}$ of the two $2N\times 2N$ matrices on the right-hand sides, i.e., for $i=m+1,m+2,\cdots,N-1$ we have

$$t_{i,i+1}(\kappa_{ij}^{\pm})^2-(\lambda_j-t_{i,i})\kappa_{ij}^{\pm}+t_{i,i-1}=0,\qquad j=1,2,\cdots,N \tag{3.7a}$$

and for $i=m,m-1,\cdots,2$ we have

$$t_{i,i-1}(\kappa_{ij}^{\pm})^2-(\lambda_j-t_{i,i})\kappa_{ij}^{\pm}+t_{i,i+1}=0,\qquad j=1,2,\cdots,N. \tag{3.7b}$$

Hence the error in the ith iteration of (3.5) is amplified by a factor of at least

$$g^{(i)}\equiv\max_{1\leqq j\leqq N}(|\kappa_{ij}^{+}|,|\kappa_{ij}^{-}|).$$

Thus, the method is expected to be stable if

$$\prod_{i=2}^{N-1}g^{(i)}=\prod_{i=2}^{N-1}\max_{1\leqq j\leqq N}(|\kappa_{ij}^{+}|,|\kappa_{ij}^{-}|)\leqq\text{Const.} \tag{3.8}$$

As a canonical example for such stable behavior, let us consider ST matrices whose entries are "slowly varying" along their diagonals, i.e., $t_{ij} \sim t_{i+1,j+1}$. Now, *if* the superdiagonal entries are properly scaled so that also $t_{i,i+1} \sim t_{i,i-1}$, then by Gershgorin estimate we have for *any* $1 \leqq j \leqq N$,

$$|\lambda_j - t_{i,i}|^2 \lesssim (|t_{i,i-1}| + |t_{i,i+1}|)^2 \lesssim 4|t_{i,i-1}| \cdot |t_{i,i+1}|,$$

and hence the product of the characteristic roots $\kappa_{ij}^{\pm}$ is of order unity, for

$$|\kappa_{ij}^{\pm}|^2 = \left|\frac{(\lambda_j - t_{i,i}) \pm \sqrt{(\lambda_j - t_{i,i})^2 - 4t_{i,i+1}t_{i,i-1}}}{2t_{i,i\pm 1}}\right|^2 \lesssim \left|\frac{t_{i,i\mp 1}}{t_{i,i\pm 1}}\right| \sim 1.$$

If, on the other hand, (3.8) fails, we have an unstable error growth at the amount $\prod_{i=2}^{N-1} g^{(i)} \gg 1$, as confirmed by the numerical examples demonstrated in § 6. Typically, such an instability shows up by the loss of orthogonality between the computed rows $p_N^{(i)}$ of P_N. Hence, one approach to solve the stability problem would be to use reorthogonalization, once the instability was detected by the loss of orthogonality; consult [3, § 3]. An alternative approach to overcome the instability problem, which better suits the proposed method, is to *restart* the recurrence relations (3.5) at the current iteration with two new successive rows of P_N. How should we obtain two such successive rows for restarting? Consider, for example, the $N = 4m$-dimensional problem. The iteration before the last of (2.4a) provides us with two $(N/2)$-dimensional unit vectors, say $z_{N/2}$ and $w_{N/2}$, where

(3.9a) $$\begin{bmatrix} z_{N/4}^{(1)} \\ z_{N/4}^{(2)} \end{bmatrix} = \begin{bmatrix} P_{N/4}^{(1)} & \vdots & \\ \cdots & \cdots & \cdots \\ & \vdots & P_{N/4}^{(2)} \end{bmatrix}^t \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \qquad \begin{bmatrix} z_{N/4}^{(1)} \\ z_{N/4}^{(2)} \end{bmatrix} \equiv \sqrt{2}\, z_{N/2},$$

(3.9b) $$\begin{bmatrix} w_{N/4}^{(1)} \\ w_{N/4}^{(2)} \end{bmatrix} = \begin{bmatrix} P_{N/4}^{(3)} & \vdots & \\ \cdots & \cdots & \cdots \\ & \vdots & P_{N/4}^{(4)} \end{bmatrix}^t \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \qquad \begin{bmatrix} w_{N/4}^{(1)} \\ w_{N/4}^{(2)} \end{bmatrix} \equiv \sqrt{2}\, w_{N/2}.$$

As before, we obtain the m and $m+1$ rows of $P_{N/2}^{(1)}$ as

(3.10a) $$\begin{aligned} p_{N/2}^{(1,m)} &= (z_{N/4}^{(1)}, O_{N/4})^t Q_{N/2}^{(1)} \\ p_{N/2}^{(1,m+1)} &= (O_{N/4}, z_{N/4}^{(2)})^t Q_{N/2}^{(1)} \end{aligned}$$

and the m and $m+1$ rows of $P_{N/2}^{(2)}$ as

(3.10b) $$\begin{aligned} p_{N/2}^{(2,m)} &= (w_{N/4}^{(1)}, O_{N/4})^t Q_{N/2}^{(2)} \\ p_{N/2}^{(2,m+1)} &= (O_{N/4}, w_{N/4}^{(2)})^t Q_{N/2}^{(2)}. \end{aligned}$$

Consequently, we can compute with $O(N^2)$ operations row numbers m, $m+1$, $3m$, and $3m+1$ of P_N, for by (2.3c) we have

(3.11) $$\begin{aligned} p_N^{(m)} &= (p_{N/2}^{(1,m)}, O_{N/2})^t Q_N \\ p_N^{(m+1)} &= (p_{N/2}^{(1,m+1)}, O_{N/2})^t Q_N \\ p_N^{(3m)} &= (O_{N/2}, p_{N/2}^{(2,m)})^t Q_N \\ p_N^{(3m+1)} &= (O_{N/2}, p_{N/2}^{(2,m+1)})^t Q_N. \end{aligned}$$

In a similar manner, one can restart the recurrence relations (3.5) at any desired iteration.

4. The eigenvectors of T_N—One at a time. In the previous section we discussed an $O(N^2)$ method for computing the whole eigensystem—eigenvalues and eigenvectors of an $N\times N$ ST matrix. In several applications one is interested in only a few of the eigenvectors of T_N. We now present a variant of this method that enables us to compute each one of the desired eigenvectors with $O(N)$ operations.

As before, we first prepare, with the help of the DC algorithm (2.4a), (2.4b), (3.4), the eigenvalues $\{\lambda^{(j)}\}_{j=1}^{N}$ and the two middle successive rows, $p_N^{(m)}$ and $p_N^{(m+1)}$ of P_N. This can be done at the expense of $O(N^2)$ operations, and in many practical cases with even less. Equipped with this we can compute the eigenvector $x_N=(x^{(1)},x^{(2)},\cdots,x^{(N)})$ corresponding to the eigenvalue, say, $\lambda^{(j)}$

$$T_N x_N=\lambda^{(j)}x_N. \tag{4.1}$$

Equation (4.1) gives us the linear three-term recurrence relations between the components of x

$$t_{i,i-1}x^{(i-1)}+t_{i,i}x^{(i)}+t_{i,i+1}x^{(i+1)}=\lambda^{(j)}x^{(i)}. \tag{4.2}$$

Since x_N coincides with the jth *column* of P_N, we have its two middle entries $x^{(m)}$ and $x^{(m+1)}$ from the jth entries of $p_N^{(m)}$ and $p_N^{(m+1)}$. The rest of the entries are computed recursively with $3N$ operations by

$$x^{(i+1)}=\frac{1}{t_{i,i+1}}[(\lambda_j-t_{i,i})x^{(i)}-t_{i,i-1}x^{(i-1)}],\qquad i=m+1,\cdots,N-1, \tag{4.3a}$$

$$x^{(i-1)}=\frac{1}{t_{i,i-1}}[(\lambda_j-t_{i,i})x^{(i)}-t_{i,i+1}x^{(i+1)}],\qquad i=m,m-1,\cdots,2. \tag{4.3b}$$

The computation is stable or unstable depending on whether

$$\prod_{i=2}^{N-1}\max(|\kappa_{ij}^{+}|,|\kappa_{ij}^{-}|) \tag{4.4}$$

is bounded or $\gg 1$.

5. Solution of the updating problem. In this section we follow [1] and [3] in a discussion of the promised $O(N^2)$ method for solving the updating problem (2.3b), i.e., computing the eigensystem of $D_N+\sigma z_N z_N^t$. Without loss of generality we may assume that $\sigma>0$ and that the problem has been deflated, so that the components of $z_N=(z^{(1)}\cdots z^{(N)})^t$, as well as the difference between any two diagonal entries of $D_N=\operatorname{diag}(d_{11}<d_{22}<\cdots<d_{NN})$, are different from zero (in practice we take a neighbourhood of zero with a preassigned tolerance, say ε); consult [1], [3], and [4, § 4]. In this case, it follows that the eigenvalues of the updating problem $\lambda^{(i)}$, $i=1,2,\cdots,N$, strictly interlace with those of D_N [1, Thm. 1], [3, Thm. 2.1]

$$d_{11}<\lambda^{(1)}<d_{22}<\lambda^{(2)}<\cdots<\lambda^{(N)}<d_{NN}+\sigma\equiv d_{N+1,N+1}. \tag{5.1}$$

With this in mind we now turn to compute the required eigenvalues $\lambda=\lambda^{(i)}$ as the roots of the so-called secular equation [6]

$$f(\lambda)\equiv 1+\sigma\sum_{j=1}^{N}\frac{(z^{(j)})^2}{d_{jj}-\lambda}=0. \tag{5.2}$$

The function $f(\lambda)$ is the rational representation of the characteristic polynomial associated with $D_N+\sigma z_N z_N^t$, and the interlacing property ensures that f has N simple roots $\lambda^{(i)}$ lying in the open intervals $(d_{ii}, d_{i+1,i+1})$, $i=1, 2, \cdots, N$. We shall mention two zero-finders that have been advocated to find these roots:

(1) The zero-finder proposed by Bunch et al. [1], which is based on rational interpolation, employs the values of $f(\lambda)$ and its first derivative, $f'(\lambda)$. The advantage of this zero-finder (which will be referred to as "zeroinder") is that it produces a *monotonic* sequence of approximations in $(d_{ii}, d_{i+1,i+1})$ that converges quadratically to $\lambda^{(i)}$. However, it is very sensitive near the ends of the intervals $(d_{ii}, d_{i+1,i+1})$, where the derivative involved, $f'(\lambda)$, becomes singular.

(2) Cuppen [3] advocated the "zeroinrat" zero-finder of Bus and Dekker [2], which is based on rational interpolation of three f-values in the interval $(d_{ii}, d_{i+1,i+1})$. This algorithm is more robust than the "zeroinder," for it does not involve $f'(\lambda)$; consequently, it avoids the previous difficulty of singular derivatives near d_{ii} and, moreover, it saves half the operations per iteration. Yet, the current "zeroinrat" algorithm lacks the monotonicity property we had before, and, therefore, it requires safeguarding to ensure that we remain within the desired interval (this decreases the convergence rate to ~ 1.839).

Assuming that either one of these zero-finders requires no more than a constant number of iterations to compute (with some preassigned tolerable accuracy) each root of (5.2), then the required eigenvalues $\lambda^{(i)}$, $i=1, 2, \cdots, N$, are obtained by $O(N^2)$ operations. Equipped with these eigenvalues, we now may use the Sherman-Morrison formula to compute the associated normalized eigenvectors, $q_N^{(i)}$, which form the column vectors of Q_N in (2.3b), as [1, § 4).

$$q_N^{(i)} = \frac{(D_N - \lambda^{(i)} I_N)^{-1} z_N}{\|(D_N - \lambda^{(i)} I_N)^{-1} z_N\|}, \qquad i = 1, \cdots, N, \tag{5.3}$$

and the total operations cost does not exceed $O(N^2)$, as asserted.

To enhance the stability properties of the whole DC algorithm, the updating problem should be solved with maximum accuracy. To achieve this, we now present an efficient implementation for the solution of this problem, based on the ingredients described above.

As a first step we reformulate (5.2) in a manner suggested in [1]. By the interlacing property (5.1) we have

$$\lambda^{(i)} = d_{ii} + \sigma \mu^{(i)}, \qquad 0 < \mu^{(i)} < 1, \qquad \sum_{i=1}^{N} \mu^{(i)} = 1. \tag{5.4}$$

For $i = 1, 2, \cdots, N$ we make the change of variables, $\lambda = d_{ii} + \sigma\mu$, so that instead of $f \equiv f(\lambda)$, we now obtain N different rational functions, $W_i(\mu)$,

$$W_i(\mu) = 1 + \sum_{j=1}^{N} \frac{(z^{(j)})^2}{\delta_{ji} - \mu}, \qquad \delta_{ji} \equiv \frac{d_{jj} - d_{ii}}{\sigma}, \qquad i = 1, 2, \cdots, N, \tag{5.5}$$

each of which has a simple root $\mu = \mu^{(i)}$ in the open interval $(\delta_{ii} \equiv 0, \delta_{i+1,i})$. Computing the root of $W_i(\mu)$ in this interval—rather than the root of $f(\lambda)$ in the $(d_{ii}, d_{i+1,i+1})$ interval—has the advantage that $W_i'(\mu)$ is *uniformly* bounded from below (by 1) rather than having $f'(\lambda) \geqq 1/\sigma$, as in [3, Thm. 3.1]. The computation of the desired root proceeds by carefully monitoring a mixture of the two zero-finders mentioned above. Namely, the "zeroinder" algorithm will be used when we are well inside the interval of interest, $(0, \delta_{i+1,i})$, while we switch to the "zeroinrat" algorithm when we approach either end of this interval. To decide upon the switching policy, we first quote the following.

LEMMA 5.1 [4, Lem. 4.6]. *Assume that the deflation test $|z^{(i)}| > \varepsilon$ is satisfied. Then either we have*

$$\frac{\varepsilon^2}{\sigma^2(2+\delta_{i+1,i})}\,\delta_{i+1,i} \leqq \mu^{(i)} \leqq \frac{1}{2}\,\delta_{i+1,i} \tag{5.6a}$$

or alternatively

$$\frac{1}{2}\,\delta_{i+1,i} \leqq \mu^{(i)} \leqq \left(1-\frac{\varepsilon^2}{\sigma^2(2+\delta_{i+1,i})}\right)\delta_{i+1,i}. \tag{5.6b}$$

The bounds on the left- and right-hand sides of (5.6) yield a closed subinterval $[L^{(i)}, H^{(i)}]$, which encloses $\mu^{(i)}$. (Experiments have shown that these bounds actually may be achieved.)

A more practical indication to the location of $\mu^{(i)}$ is obtained from the following considerations. The rational function

$$V_i(\mu) = \text{Const}_i + \frac{(z^{(i)})^2}{-\mu} + \frac{(z^{(i+1)})^2}{\delta_{i+1,i}-\mu}, \qquad \text{Const}_i \equiv \sum_{j\neq i,i+1} \frac{(z^{(j)})^2}{\delta_{ji}-\delta_{i+1,i}} \tag{5.7a}$$

has a simple root, $l^{(i)}$, in the interval $(0, \delta_{i+1,i})$. Since $V_i(\mu)$ dominates $W_i(\mu)$ in that interval and they are both monotonically increasing, we can use this root (that is found by solving a simple quadratic equation) as a lower bound for $\mu^{(i)}$. Similarly, the function

$$U_i(\mu) = \text{Const}_i + \frac{(z^{(i)})^2}{-\mu} + \frac{(z^{(i+1)})^2}{\delta_{i+1,i}-\mu}, \qquad \text{Const}_i \equiv \sum_{j\neq i,i+1} \frac{(z^{(j)})^2}{\delta_{ji}} \tag{5.7b}$$

has a simple root $h^{(i)}$ in the interval $(0, \delta_{i+1,i})$, which may serve as an upper bound for $\mu^{(i)}$.

Returning to our problem of finding the roots of $W_i(\mu)$ in (5.5), we use the "zeroinder" algorithm when inside the $(0, \delta_{i+1,i})$ interval. This requires us to compute $W_i'(\mu)$, and Lemma 5.1 indicates that as we approach either end of the interval, the computation of $W_i'(\mu)$ involves factors of ε^{-4} that will lead to an underflow problem. To avoid this situation, we use a switching policy, which in each step tests if either one of the following inequalities is satisfied:

$$l^{(i)} \leqq L^{(i)}, \qquad h^{(i)} \geqq H^{(i)}, \qquad h^{(i)} \leqq l^{(i)} \tag{5.8}$$

as an indication that we are in the neighborhood of the singular ends, in which case we use the "zeroinrat" algorithm instead. This "switching" policy enabled us to achieve, with the usual 64-bit arithmetic, more than satisfactory results that otherwise would have required the less attractive extended precision arithmetic.

Concerning the computation of the eigenvectors in (5.3), we note that it is possible to have severe round-off when $\lambda^{(i)}$ is close to d_{ii} or $d_{i+1,i+1}$ [3, § 2]. The reformulation of the eigenvalue problem in (5.5) enables one to avoid half of these round-off problems, namely, when $\lambda^{(i)}$ is close to d_{ii}. Indeed, the normalized eigenvectors, $q_N^{(i)}$, are now given by

$$q_N^{(i)} = \frac{[D_N^{(i)}]^{-1} z_N}{\|[D_N^{(i)}]^{-1} z_N\|}, \qquad D_N^{(i)} = \text{diag}\,(\delta_{1i} \cdots \delta_{N_i}) - \mu^{(i)} I. \tag{5.9}$$

Using (5.9) instead of (5.3) avoids cancellation that arises when $\lambda^{(i)}$ is too close to d_{ii}, i.e., when $\mu^{(i)}$ is too close to zero, for $\delta_{ii} \equiv 0$ in this case. We are still left with the other half of the cancellation problem when $\lambda^{(i)}$ is too close to $d_{i+1,i+1}$. If this is indeed the case (as we can foresee by computing the practical bounds for $\mu^{(i)}$ from (5.7a), (5.7b)), then we propose to perform yet another reformulation of our eigenvalue problem, using

$$(5.10)\qquad \lambda^{(i)} = d_{i+1,i+1} - \sigma\eta^{(i)}$$

instead of (5.4). In this case the role of the rational functions $W_i(\mu)$ in (5.5) is played by

$$(5.11)\qquad \bar{W}_i(\eta) = 1 + \sum_{j=1}^{N} \frac{(z^{(j)})^2}{\delta_{j,i+1} - \eta}, \qquad \delta_{j,i+1} = \frac{d_{jj} - d_{i+1,i+1}}{\sigma}, \qquad i = 1, 2, \cdots, n,$$

each of which has a simple root $\eta = \eta^{(i)}$ in the open interval $(0, -\delta_{i,i+1})$. The corresponding normalized eigenvectors are given by

$$(5.12)\qquad q_N^{(i)} = \frac{[\bar{D}_N^{(i)}]^{-1} z_N}{\|[\bar{D}_N^{(i)}]^{-1} z_N\|}, \qquad \bar{D}_N^{(i)} = \operatorname{diag}(\delta_{1,i+1} \cdots \delta_{N,i+1}) + \eta^{(i)} I_N,$$

and cancellation that arises when $\lambda^{(i)}$ is too close to $d_{i+1,i+1}$, i.e., when $\eta^{(i)}$ is too close to zero is avoided for $\delta_{i+1,i+1} \equiv 0$.

6. Numerical experiments. The main object of our experiments was a comparison between the standard $O(N^3)$ DC algorithm for computing the eigensystems of $N \times N$ ST matrices (2.3a)-(2.3c), and the proposed $O(N^2)$ method in (2.4a), (2.4b), and (3.4), which makes use of the three-term recurrence relations (3.5a), (3.5b). The input data for these relations, the eigenvalues $\lambda^{(i)}$ and the two initial successive row vectors $p_N^{(m)}$, $p_N^{(m+1)}$ were supplied with maximum accuracy, with the help of the updating solver described in § 5 that avoids extended precision. Indeed all our calculations, including the pathologically ill-conditioned W_{+N}- Wilkinson's matrices, were carried out with a 64-bit arithmetic.

The first set of results includes "well-behaved" matrices taken from [7, (7.4)-(7.9)]. The entries along the diagonals of these matrices are "slowly varying" and their eigenvalues are equally distributed. The stability analysis in § 3 indicates bounded amplification factors in these cases, and the numerical results confirmed the expected stable behavior of our method. Table 1 summarizes the results for the prototype ST matrix of this group where $T_{ij} = 2 - |i - j|$.

Since the rows of P were constructed by equating to zero rows $2, 3, \cdots, N-1$ of $TP - P\Lambda$, the quantities on the left columns, $\|TP - P\Lambda\|_\infty$, stand for

$$\max(\|t_{1,1}p^{(1)} + t_{1,2}p^{(2)} - p^{(1)}\Lambda\|, \|t_{N,N-1}p^{(N-1)} + t_{N,N}p^{(N)} - p^{(N)}\Lambda\|).$$

They may serve us as a quantitive indication of the accumulation of rounding errors in the three-term recursion relations (3.5), which is responsible for the loss of (no more than) two orders of magnitude relative to the standard algorithm. The advantage of the proposed method lies upon the fact that the results on the right columns are

TABLE 1
Results for $T[1, 2, 1]$ *matrix of order N.*

	Standard DC algorithm		The proposed method	
N	$\|TP-P\Lambda\|_\infty$	$\|P'P-I\|_\infty$	$\|TP-P\Lambda\|_\infty$	$\|P'P-I\|_\infty$
101	2.5E−15	6.2E−16	9.5E−15	7.2E−15
201	2.6E−15	2.5E−15	2.2E−14	1.5E−14
301	3.0E−15	2.8E−15	2.9E−14	8.8E−14
401	4.0E−15	6.9E−15	2.5E−13	1.2E−13

obtained by saving order of magnitude in execution time relative to the results on the left columns.

Next, we turn to the second group of matrices that consists of Wilkinson's matrices, W_{+N}. The superdiagonals of these matrices are properly scaled to begin with—they all equal one; the entries along the main diagonal, however, diag $((N-1)/2, (N-3)/2, \cdots, 1, 0, 1, \cdots, (N-1)/2)$ are far from being "slowly varying." This leads to amplification factors of the recurrence relations (3.5) of order $\sim(N-1)/2!$, which indicates loss of all (64-bit precision) significant figures in computing the eigenproblem of W_{+N} of order $N \gtrsim 40$. Moreover, the largest eigenvalues of W_{+N} are clustered in pairs, which may be inseparable up to the 14th decimal digit. This then leads to additional inaccuracies in the updating solution (while seeking two extremely close roots of the secular equation) as well as in the deflation process. As a result, the initial input data for the recurrence relation will also suffer from loss of accuracy. These arguments are well reflected in Table 2.

In order to be competitive with the standard algorithm that gave excellent results for W_{+N} up to order $N=201$, an attempt was made to improve the results of our method. To this end we have appended our method with the *restarting* procedure described in § 3. Thus, by computing the row vectors (here $m=(N+1)/4$) $p_N^{(m)}, p_N^{(m+1)}$, $p_N^{(3m)}$, and $p_N^{(3m+1)}$ as additional input data to restart the three-term recurrence relations, we were able to get decent results for the W_{+N}-matrices up to order $N \sim 200$. Repeating such restarting procedures would enable us to deal with even larger W_{+N}-matrices, still within the $O(N^2)$ operations limit.

Finally, the last group of matrices that were tested consists of randomly generated entries in $[-1, 1]$. The results obtained are summarized in Table 3.

We observe that excellent results are obtained by our method for such randomly generated matrices of order up to $N \sim 100$. If additional restarting procedures were employed every 100-200 iterations, it would enable us to achieve highly accurate results for matrices of almost any practical size.

TABLE 2
Results for the W_{+N} matrices.

	Standard DC algorithm		The proposed method	
N	$\|TP-P\Lambda\|_\infty$	$\|P'P-I\|_\infty$	$\|TP-P\Lambda\|_\infty$	$\|P'P-I\|_\infty$
21	4.5E−16	2.5E−16	1.2E−12	9.8E−10
41	1.3E−15	9.4E−16	3.7E−8	7.4E−7
47	2.0E−15	9.1E−16	5.3E−3	1.0E−3
49	2.0E−15	9.8E−16	0.12	0.23

TABLE 3
Results for random matrices of order N.

	Standard DC algorithm		The proposed method	
N	$\|TP-P\Lambda\|_\infty$	$\|P'P-I\|_\infty$	$\|TP-P\Lambda\|_\infty$	$\|P'P-I\|_\infty$
100	8.4E−15	9.8E−16	9.5E−15	7.6E−15
200	5.9E−15	3.4E−15	6.2E−9	9.8E−8
300	6.3E−15	5.6E−15	4.2E−4	3.1E−2
400	7.2E−15	6.8E−15	$O(1)$	$O(1)$

In summary, we conclude that the proposed method for solving the eigenproblem of ST matrices provides a competitive alternative to the standard eigensolvers for a certain class of such matrices; by sacrificing a *few* orders of accuracy, the method enables one to save *order of magnitude* in the total execution time. This conclusion was confirmed by further extensive numerical experiments reported in [5].

Acknowledgment. We thank Professor Beresford Parlett for an enlightening discussion on the Inverse Iteration method.

REFERENCES

[1] J. R. BUNCH, C. P. NEILSEN, AND D. C. SORENSEN, *Rank one modification of the symmetric eigenproblem*, Numer. Math., 31 (1978), pp. 31-48.

[2] J. C. BUS AND T. J. DEKKER, *Two efficient algorithms with guaranteed convergence for finding a zero of a function*, TOMS 1, 1975, pp. 330-345.

[3] J. J. M. CUPPEN, *A divide and conquer method for the symmetric tridiagonal eigenproblem*, Numer. Math., 36 (1981), pp. 177-195.

[4] J. J. DONGARRA AND D. C. SORENSEN, *A fully parallel algorithm for the symmetric eigenvalue problem*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 139-154.

[5] D. GILL, *Divide and conquer—A parallel algorithm for computing the spectral decomposition of symmetric matrices*, M.Sc. thesis, Tel-Aviv University, Tel-Aviv, Israel, 1987.

[6] G. H. GOLUB, *Some modified matrix eigenvalue problems*, SIAM Rev., 15 (1973), pp. 318-334.

[7] R. T. GREGORY AND D. L. KARNEY, *A Collection of Matrices for Testing Computational Algorithms*, John Wiley, New York, 1969.

[8] B. N. PARLETT, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[9] G. PETERS AND J. H. WILKINSON, *Eigenvalues of* $Ax = \lambda Bx$ *with band symmetric* A *and* B, Comput. J., 12 (1969), pp. 398-404.

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

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 1, pp. 174-179, January 1990

RELATIVE SIZES OF THE HESSIAN TERMS IN NONLINEAR PARAMETER ESTIMATION*

J. M. VARAH†

Abstract. The computational estimation of parameters in nonlinear models leads to nonlinear least squares problems with Hessians having a particular structure: $H = J^TJ + B$, with J the Jacobian matrix and B arising from second derivative terms. In practice, it has been noticed that B is very often considerably smaller than J^TJ, particularly near a local minimum, so that H can often be well approximated by the first derivative term. In this paper an attempt is made to give some explanation for this behaviour and to indicate how to check for it during the computational process.

Key words. parameter estimation, nonlinear least squares

AMS(MOS) subject classification. 65F20

1. Introduction. The least squares fitting of parameters $\mathbf{p} = (p_1, \cdots, p_m)$ to nonlinear models, either using a functional model

$$y = f(t; \mathbf{p}) \tag{1.1}$$

or a differential equation model

$$y' = f(t; \mathbf{p}), \tag{1.2}$$

with values specified at certain points $y(t_i) = y_i$, $i = 1, \cdots, n$, leads to the minimization of a discrete nonlinear least squares function

$$\phi(\mathbf{p}) = \sum_{i=1}^{n} (y(t_i; \mathbf{p}) - y_i)^2 \equiv \sum_{1}^{n} d_i^2. \tag{1.3}$$

Suppose $\mathbf{p}^*$ is a local minimum for $\phi(\mathbf{p})$. Then in a neighbourhood of $\mathbf{p}^*$, we have

$$y(t_i; \mathbf{p}^* + \delta_p) = y(t_i; \mathbf{p}^*) + (J\delta_p)_i + \tfrac{1}{2}(\delta_p^T W^{(i)} \delta_p) + O(\|\delta_p\|^3)$$

where J is the $n \times m$ Jacobian matrix

$$J_{ij} = \frac{\partial}{\partial p_j} [y(t_i; \mathbf{p}^*)]$$

and $\{W^{(i)}\}_1^n$ are the $m \times m$ second derivative matrices

$$W_{jk}^{(i)} = \frac{\partial^2 y(t_i; \mathbf{p}^*)}{\partial p_j \partial p_k}.$$

Since the gradient of $\phi(\mathbf{p}^*)$, $\mathbf{g}(\mathbf{p}^*) = J^T\mathbf{d} = 0$, the behaviour of $\phi(\mathbf{p})$ near $\mathbf{p}^*$ is as follows:

$$\phi(\mathbf{p}^* + \delta_p) = \phi(\mathbf{p}^*) + \delta_p^T H \delta_p + O(\|\delta_p\|^3) \tag{1.4}$$

where the Hessian matrix H has two parts:

$$H = J^TJ + \sum_{i=1}^{n} d_i W^{(i)} \equiv J^TJ + B. \tag{1.5}$$

The nature of the (positive definite) matrix $H(\mathbf{p}^*)$ is crucial to the understanding of the parameter estimation problem at hand. In particular,

* Received by the editors September 26, 1988; accepted for publication (in revised form) April 14, 1989.

† Computer Science Department, University of British Columbia, Vancouver, British Columbia V6T 1W5, Canada.

(i) The convergence of numerical schemes for minimizing $\phi(\mathbf{p})$ depends on the nature of H and on approximating it accurately (see Dennis and Schnabel (1983)).

(ii) The sensitivity of the parameter values $\mathbf{p}^*$ depends on the near-singularity of H, and hence on accurate determination of its condition number (see Varah (1987)).

(iii) In a statistical context, after making appropriate assumptions about the nature of the errors involved, the resulting confidence regions for the parameters will depend on H (see Draper and Smith (1981)). Moreover, decisions on the viability of linearized confidence regions can be made on the basis of H and its constituent parts (see Bates and Watts (1980) or Ratkowsky (1983)).

In practice, when actually computing parameter values, it has been noticed that at or near the local minimum $\mathbf{p}^*$, H is often well approximated by J^TJ. Of course, from (1.5), the B matrix will be small when either the residuals $\mathbf{d}$ or when the second derivative matrices $W^{(i)}$ are small. However, the phenomenon appears to occur more generally; for example, Donaldson and Schnabel (1987), in an extensive study of linear versus nonlinear confidence region for algebraic problems (1.1), found very little difference in the regions generated using H or J^TJ in a wide variety of examples. This has also been the present author's experience in dealing with differential equation models (1.2).

From a computational point of view, it is much easier to monitor and approximate the Jacobian matrix J than the second derivative matrices $W^{(i)}$. Thus if it were known that H was well approximated by J^TJ, there would be a large saving in computational effort.

It is the purpose of this note to provide some explanation for this behaviour. In § 2, we cast the problem into the more convenient continuous least squares setting and identify one attribute of a given model that will lead to the above behaviour. In § 3, we investigate the relationship with the recent statistical work involving curvature measures of nonlinearity. Finally, in § 4 we present some numerical examples illustrating the behaviour.

2. Projecting the second derivative matrix. Although we could continue our discussion using the discrete nonlinear least squares problem given above, we feel it is more convenient (and more transparent) first to use the corresponding continuous problem: given a model $y=f(t;\mathbf{p})$ and a data function $\hat{y}(t)$ in some interval, find $\mathbf{p}^*$ to (locally) minimize

$$\theta(\mathbf{p})=\int (y(t;\mathbf{p})-\hat{y}(t))^2\,dt. \tag{2.1}$$

If we let $z^{(j)}(t)=\partial y/\partial p_j$ and $W_{jk}(t)=\partial^2 y/\partial p_j\,\partial p_k$, then the gradient of θ,

$$\frac{\partial\theta}{\partial p_j}=g_j(\mathbf{p})=\int (y(t;\mathbf{p})-\hat{y}(t))z^{(j)}(t)\,dt \qquad (=0 \text{ for } \mathbf{p}=\mathbf{p}^*)$$

and the Hessian

$$H_{jk}=\frac{\partial^2\theta}{\partial p_j\,\partial p_k}=\int z^{(j)}(t)z^{(k)}(t)\,dt+\int (y(t;\mathbf{p})-\hat{y}(t))\,W_{jk}\,dt\equiv A_{jk}+B_{jk}. \tag{2.2}$$

Typically, a given discrete problem (1.3) will be not unlike its continuous analogue, particularly with a fairly large number of discrete data points. Note that the A-matrix is the Gram matrix of the functions $\{z^{(j)}(t)\}_1^m$.

The natural function space for this continuous least squares problem is the space of square integrable functions over the given interval, with $\|f\|_2^2=\int f(t)^2\,dt$. We assume for the given problem that all the first and second derivative functions appearing above are smooth and square integrable.

Now let S be the subspace spanned by $\{z^{(j)}(t)\}_1^m$, and for each j, k, let $\hat{W}_{jk}(t)$ be the projection of $W_{jk}(t)$ into S. If $\{q^{(1)},\cdots,q^{(m)}\}$ forms an orthonormal basis for S, then

$$\hat{W}_{jk}(t)=\sum_{i=1}^m b_{jk}^{(i)}q^{(i)}(t),\qquad b_{jk}^{(i)}=\int W_{jk}(t)q^{(i)}(t)\,dt,$$

and

$$\|\hat{W}_{jk}\|_2^2=\sum_{i=1}^m (b_{jk}^{(i)})^2,\qquad \|W_{jk}-\hat{W}_{jk}\|_2^2=\|W_{jk}\|_2^2-\|\hat{W}_{jk}\|_2^2.$$

The crucial observation is the following: in the evaluation of the B-matrix in (2.2), at a local minimum $\mathbf{p}=\mathbf{p}^*$, the term involving the projection $\hat{W}_{jk}(t)$ is zero. In fact, we have the following:

THEOREM 2.1. *Consider the Hessian matrix H evaluated at $\mathbf{p}=\mathbf{p}^*$ as in* (2.2). *For each j, $k=1,\cdots,m$, let $\hat{W}_{jk}(t)$ be decomposed as above. Then*

$$|B_{jk}|^2\leqq\theta(\mathbf{p}^*)\|W_{jk}-\hat{W}_{jk}\|_2^2.$$

Proof. We have

$$\begin{aligned}B_{jk}&=\int(y-\hat{y})W_{jk}\,dt\\&=\int(y-\hat{y})\hat{W}_{jk}\,dt+\int(y-\hat{y})(W_{jk}-\hat{W}_{jk})\,dt\\&=0+\int(y-\hat{y})(W_{jk}-\hat{W}_{jk})\,dt.\end{aligned}$$

Thus

$$\begin{aligned}|B_{jk}|^2&\leqq\int(y-\hat{y})^2\,dt\cdot\int(W_{jk}-\hat{W}_{jk})^2\,dt\\&=\theta(\mathbf{p}^*)\|W_{jk}-\hat{W}_{jk}\|_2^2.\end{aligned}$$ □

Hence not only will the elements of B be small when the residual is small or when the second derivative terms are small, but at a local minimum, they will be small if the second derivative terms "look like" the first derivatives. In practice, this seems to be a common occurrence, which we illustrate in the examples.

In the discrete setting of § 1, with data points $\{t_i\}_1^n$, the procedure is basically the same, except that the norm is now the l_2 vector norm. For the set of second derivative matrices $\{W^{(i)}\}_1^n$, we form for each j, k the projection vector $\hat{W}_{jk}$ in the space S spanned by the columns of the Jacobian matrix. If $J=QR$ is the QR decomposition of $J(\mathbf{p}^*)$, then

$$\hat{W}_{jk}=\sum_{i=1}^m b_{jk}^{(i)}q^{(i)}$$

where the matrices

$$B^{(i)} = \sum_{i=1}^{n} q_j^{(i)} W^{(l)}.$$

As in the continuous case, we have

$$\|\hat{W}_{jk}\|_2^2 = \sum_{i=1}^{m} (b_{jk}^{(i)})^2, \qquad \|W_{jk} - \hat{W}_{jk}\|_2^2 = \|W_{jk}\|_2^2 - \|\hat{W}_{jk}\|_2^2. \tag{2.3}$$

And similarly we obtain the following theorem.

THEOREM 2.2. *Consider the discrete Hessian matrix* $H = J^TJ + B$ *from* (1.5), *evaluated at* $\mathbf{p} = \mathbf{p}^*$. *For each* $j, k = 1, \cdots, m$, *let* W_{jk} *be decomposed as above. Then*

$$|B_{jk}|^2 \leqq \phi(\mathbf{p}^*)\|W_{jk} - \hat{W}_{jk}\|_2^2 \equiv C_{jk}. \tag{2.4}$$

In the discrete case, it may also be of interest to compare the relative sizes of the two Hessian components. Using $J = \mathrm{QR}$, we can write

$$H = \hat{R}^T(I + \hat{R}^{-T}B\hat{R}^{-1})\hat{R},$$

where $\hat{R}$ consists of the (nonzero) first m rows of R. The marix $\tilde{B} = \hat{R}^{-T}B\hat{R}^{-T}$ is sometimes called the curvature matrix (see Björck (1988)). When the Jacobian is ill conditioned, this matrix may be much larger than B. One effect of this is that H may "look like" J^TJ in absolute terms, but its small eigenvalues may be significantly different from the small eigenvalues of J^TJ in relative terms.

Computationally, it is worth mentioning that $\|\hat{W}_{jk} - W_{jk}\|_2$ can be evaluated using (2.3) without storing all the columns of Q. After computing the $\{W^{(i)}\}$ matrices, one merely needs to compute $B^{(1)}, \cdots, B^{(m)}$ and $\|\hat{W}_{jk}\|_2$ as above.

Thus one can bound the size of the B matrix in a practical situation by using Theorem 2.2. In the examples given in § 4, the actual bounds computed give quite accurate estimates of the size of B.

3. Relation to statistical curvatures.

There has been a great deal of discussion in the statistical literature concerning the effects of nonlinearity in parameter models like (1.1). For example, the question of validity of linearized confidence regions in a nonlinear model is very important. The first classic paper in this area is Beale (1960), where the author attempts to quantify the effect of the nonlinearity in the model. More recently, Bates and Watts (1980) introduced the idea of relative curvatures and related them to Beale's work. This paper has spawned a great deal of further investigation (see, for example, Ratkowsky (1983) and Cook and Goldberg (1986)).

These relative curvatures can be described using the procedure of the previous section, applied in the discrete setting. The projection vectors $\hat{W}_{jk}$ are computed for each j, k, producing "tangential" and "normal" component vectors

$$\hat{W}_{jk} \quad \text{and} \quad \tilde{W}_{jk} = W_{jk} - \hat{W}_{jk}.$$

These vectors are then recast in matrix form as $\{\hat{W}^{(i)}\}_1^n$ and $\{\tilde{W}^{(i)}\}_1^n$, and the relative curvatures defined as

$$\hat{\Gamma} = s\sqrt{m} \cdot \max_{\mathbf{u}\in R^m} \frac{\|\hat{\mathbf{v}}(\mathbf{u})\|_2}{\|J\mathbf{u}\|_2^2} \qquad \text{(parameter-effects curvature)},$$

$$\tilde{\Gamma} = s\sqrt{m} \cdot \max_{\mathbf{u}\in R^m} \frac{\|\tilde{\mathbf{v}}(\mathbf{u})\|_2}{\|J\mathbf{u}\|_2^2} \qquad \text{(intrinsic curvature)}.$$

Here $s^2 = \phi(\mathbf{p}^*)/(n-m)$ and $\hat{\mathbf{v}}_i(\mathbf{u}) = \mathbf{u}^T\hat{W}^{(i)}\mathbf{u}$, $\tilde{\mathbf{v}}_i(\mathbf{u}) = \mathbf{u}^T\tilde{W}^{(i)}\mathbf{u}$.

Note that although $\{\tilde{W}^{(i)}\}$ will be small when the second derivatives look like the first, this may not be reflected in $\tilde{\Gamma}$ being small if J is ill conditioned.

4. Numerical examples. Here we present four fairly representative models, each of which has been used in the numerical or statistical literature. For the purposes of this paper, we generate data for each example by making random perturbations to an exact model (where $\mathbf{p}=\mathbf{p}_e$) at a prescribed noise level ε. The results given are representative for the models generally; in all cases we use 10 data points and $0\leqq t\leqq 3$ as the range of the independent variable, but the results do not change significantly for other choices. Since the models are nonlinear, the results do vary with the choice of parameters $\mathbf{p}$, and where appropriate we have included two values for $\mathbf{p}_e$. The more interesting results are for relatively high noise levels, and we include results for $\varepsilon=10^{-2}$ and 10^{-1}.

For each example and each data set we minimize $\phi(\mathbf{p})$, obtaining $\mathbf{p}^*$, and then compute matrices, J, B, H, and $\{W^{(i)}\}$. We then compute $\tilde{B}$ and the size of the projection $\hat{W}$. We indicate the sizes of matrices B, $\tilde{B}$, and $\|\hat{W}-W\|_2$ by their maximum elements, and the conditioning of J and H by the minimum eigenvalues $\lambda_1(J^TJ)$ and $\lambda_1(H)$. The maximum elements of J, H, and W are of order unity. We also give $C_{\max}=\max C_{ij}$, our bound for B computed from (2.4).

Example 1 (Michaelis-Menten model). $y(t;p)=p_1t/(p_2+t)$.
This is a common example in the statistical literature; see, for example, Bates and Watts (1980) or Cook and Goldberg (1986). Refer to Table 1.

The first choice of $\mathbf{p}_e$ is typical for the example: the problem is reasonably well conditioned, and B is three or four orders of magnitude smaller than H or J. The second choice is included because it is comparable to the data used by Bates and Watts. This case is much more poorly conditioned, which is (at least partly) due to poor scaling of J for these parameter values. The poor conditioning is reflected in the large difference between $\mathbf{p}_e$ and $\mathbf{p}^*$.

Example 2 (Two exponential model). $y(t;\mathbf{p})=p_1e^{p_2t}+p_3e^{p_4t}$.

This is a well-known, ill-conditioned model from the numerical literature (see, for example, Varah (1985)). The results are similar to those of Example 1, except that this example is more uniformly ill conditioned. Notice also that here $\tilde{B}$ is of order unity, but B is of order 10^{-3} or 10^{-4}. Refer to Table 2.

Example 3 (Asymptotic regression model). $y(t;\mathbf{p})=p_1+p_2e^{p_3t}$.

This well-conditioned model is standard in the statistical literature (see Ratkowsky (1983), p. 101 or Donaldson and Schnabel (1987)). Even in this example, although

TABLE 1

$\mathbf{p}_e$	ε	$\mathbf{p}^*$	$\sqrt{\phi(\mathbf{p}^*)}$	$\lvert B\rvert_{\max}$	$\lambda_1(J^TJ)$	$\lambda_1(H)$	$\lvert\tilde{B}\rvert_{\max}$	$\|\hat{W}-W\|_{\max}$	$C_{\max}$
1.0, 1.0	10^{-2}	1.007, 1.02	0.016	1.2×10^{-4}	0.032	0.032	3.0×10^{-3}	0.056	9.0×10^{-4}
1.0, 1.0	10^{-1}	1.09, 1.09	0.107	2.3×10^{-3}	0.031	0.029	0.06	0.052	5.6×10^{-3}
0.1, 1.7	10^{-2}	0.097, 1.61	0.019	1.6×10^{-5}	8.5×10^{-5}	1.0×10^{-4}	0.19	1.6×10^{-3}	3.0×10^{-5}
0.1, 1.7	10^{-1}	0.018, 2.26	0.131	6.3×10^{-6}	9.0×10^{-7}	7.2×10^{-6}	7.0	1.0×10^{-4}	1.3×10^{-5}

TABLE 2

$\mathbf{p}_e$	ε	$\mathbf{p}^*$	$\sqrt{\phi(\mathbf{p}^*)}$	$\lvert B\rvert_{\max}$	$\lambda_1(J^TJ)$	$\lambda_1(H)$	$\lvert\tilde{B}\rvert_{\max}$	$\|\hat{W}-W\|_{\max}$	$C_{\max}$
1, −1, 1, −2	10^{-2}	0.97, −0.98	0.013	2.7×10^{-4}	1.3×10^{-4}	1.6×10^{-4}	0.19	0.024	3.1×10^{-4}
		1.03, −2.03							
1, −1, 1, −2	10^{-1}	0.84, −0.85	0.130	4.1×10^{-3}	6.4×10^{-4}	1.3×10^{-3}	1.10	0.045	5.9×10^{-3}
		1.24, −2.31							

TABLE 3

$\mathbf{p}_e$	ε	$\mathbf{p}^*$	$\sqrt{\phi(\mathbf{p}^*)}$	$\lvert B\rvert_{\max}$	$\lambda_1(J^TJ)$	$\lambda_1(H)$	$\lvert\tilde{B}\rvert_{\max}$	$\lVert\hat{W}-W\rVert_{\max}$	$C_{\max}$
1, 1, −1	10^{-2}	1.007, 0.995, −1.01	0.010	5.4×10^{-4}	0.088	0.089	5.6×10^{-3}	0.15	1.5×10^{-3}
1, 1, −1	10^{-1}	0.997, 1.02, −1.008	0.19	1.7×10^{-2}	0.093	0.078	0.16	0.16	3.0×10^{-2}
1, −1, −1	10^{-2}	1.0, −0.998, −1.004	0.016	1.2×10^{-3}	0.089	0.090	0.012	0.16	2.6×10^{-3}
1, −1, −1	10^{-1}	0.94, −0.95, −1.03	0.13	8.3×10^{-3}	0.082	0.089	0.094	0.15	2.0×10^{-2}

TABLE 4

$\mathbf{p}_e$	ε	$\mathbf{p}^*$	$\sqrt{\phi(\mathbf{p}^*)}$	$\lvert B\rvert_{\max}$	$\lambda_1(J^TJ)$	$\lambda_1(H)$	$\lvert\tilde{B}\rvert_{\max}$	$\lVert\hat{W}-W\rVert_{\max}$	$C_{\max}$
1, 1, 1, 1, 1	10^{-2}	1.01, 1.02, 0.996, 1.02 1.04	0.013	2.7×10^{-4}	6.9×10^{-4}	6.7×10^{-4}	0.026	0.024	3.1×10^{-4}
1, 1, 1, 1, 1	10^{-1}	0.22, 0.77 1.22, 0.68, 1.56	0.12	1.0×10^{-3}	1.8×10^{-4}	5.4×10^{-4}	1.9	0.014	1.7×10^{-3}
0.6, −0.7 −0.4, 1.5, 0.1	10^{-2}	0.588, −0.695, −0.388, 1.49, 0.068	0.010	2.9×10^{-3}	0.038	0.040	0.052	0.89	8.9×10^{-3}

$\|\hat{W}-W\|$ is not overly small, the actual size of B is still two orders of magnitude smaller than J or H. Refer to Table 3.

Example 4 (Bent-hyperbola model). $y(t;\mathbf{p})=p_1+p_2(t-p_4)+p_3[(t-p_4)^2+p_5]^{1/2}$. This model appears in Ratkowsky (1983), p. 122, and is used by Donaldson and Schnabel (1987). For this example, the conditioning depends significantly on the location of the parameters $\mathbf{p}$, but again the elements of B are uniformly small. Note: for the last choice of $\mathbf{p}_e$, we were unable to obtain convergence for noise level $\varepsilon = 10^{-1}$, as the last parameter p_5 insisted on straying into the negative region during the search. Refer to Table 4.

REFERENCES

D. M. BATES AND D. G. WATTS (1980), *Relative curvature measures of nonlinearity*, J. Royal Statist. Soc. Ser. B, 42, pp. 1-16.

E. M. L. BEALE (1960), *Confidence regions in nonlinear estimation*, J. Royal Statist. Soc. Ser. B, 22, pp. 41-76.

A. BJÖRCK (1988), *Least squares methods*, in Handbook of Numerical Analysis, P. Ciarlet and J. Lions, eds., to appear.

R. D. COOK AND M. L. GOLDBERG (1986), *Curvatures for parameter subsets in nonlinear regression*, Ann. Statist., 14, pp. 1399-1418.

J. DENNIS AND R. SCHNABEL (1983), *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, New York.

J. R. DONALDSON AND R. B. SCHNABEL (1987), *Computational experience with confined regions and confidence intervals for nonlinear last squares*, Technometrics, 29, pp. 67-82.

N. R. DRAPER AND H. SMITH (1981), *Applied Regression Analysis* II, John Wiley, New York.

D. A. RATKOWSKY (1983), *Nonlinear Regressing Modelling*, Marcel Dekker, New York.

J. M. VARAH (1985), *On fitting exponentials by nonlinear least squares*, SIAM J. Sci. Statist. Comp, 6, pp. 30-44.

——— (1987), *On the condition of parameter estimation problems*, in Proc. NPL Conference on Advances in Reliable Numerical Computation, Teddington, England, July 1987.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 1, pp. 180-201, January 1990

COMPUTING A TRUST REGION STEP FOR A PENALTY FUNCTION*

THOMAS F. COLEMAN†‡ AND CHRISTIAN HEMPEL†

Abstract. The problem of minimizing a quadratic function subject to an ellipsoidal constraint when the matrix involved is the Hessian of a quadratic penalty function (i.e., a function of the form $p(x)=f(x)+(1/2\mu)c(x)^Tc(x))$ is considered. Most applications of penalty functions require $p(x)$ to be minimized for values of μ decreasing to zero. In general, as μ tends to zero the nature of finite precision arithmetic causes a considerable loss of information about the null space of the constraint gradients when $\nabla^2p(x)$ is formed. This loss of information renders ordinary trust region Newton's methods unstable and degrades the accuracy of the solution to the trust region problem. The algorithm of Moré and Sorensen [*SIAM J. Sci. Statist. Comput.*, 4 (1983), pp. 553-572] is modified so as to be more stable and less sensitive to the nature of finite precision arithmetic in this situation. Numerical experiments clearly demonstrate the stability of the proposed algorithm.

Key words. equality constrained optimization, trust region, penalty function, extended system

AMS(MOS) subject classifications. 49D30, 49D37, 65K05, 65K10

1. Introduction.

The generic trust region problem is

$$\text{minimize} \quad \psi(s)=\tfrac{1}{2}s^THs+g^Ts, \qquad \text{subject to} \quad \|s\|\leqq\Delta \tag{1.1}$$

where Δ is a positive number, $g\in R^n$, and $H\in R^{n\times n}$ is a symmetric matrix. We will concern ourselves with the case where H is the Hessian of an L_2 penalty function. That is, H is the Hessian of a function of the form

$$p(x)=f(x)+\frac{1}{2\mu}c(x)^Tc(x) \tag{1.2}$$

where $f: R^n\rightarrow R$, $c: R^n\rightarrow R^t$ for $t\leqq n$, and $\mu>0$. For such functions the quadratic $\psi(s)$ is meant to serve as a local model at a current point x_c. Evaluating the first and second derivatives of $p(x)$ at x_c gives

$$g=\nabla p(x_c)=\nabla f(x_c)+\frac{1}{\mu}\nabla c(x_c)c(x_c) \tag{1.3}$$

and

$$H=\nabla^2p(x_c)=\nabla^2f(x_c)+\frac{1}{\mu}\sum_{i=1}^{t}\nabla^2c_i(x_c)c_i(x_c)+\frac{1}{\mu}\nabla c(x_c)\nabla c(x_c)^T. \tag{1.4}$$

If we define $A=\nabla c(x_c)$ and

$$B=\nabla^2f(x_c)+\frac{1}{\mu}\sum_{i=1}^{t}\nabla^2c_i(x_c)c_i(x_c), \tag{1.5}$$

then

$$H=B+\frac{1}{\mu}AA^T. \tag{1.6}$$

* Received by the editors August 5, 1987; accepted for publication (in revised form) November 18, 1988. This research was supported by the Applied Mathematical Sciences Research Program (KC-04-02) of the Office of Energy Research of the U.S. Department of Energy under grant DE-FG02-86ER25013.A000.

† Center for Applied Mathematics, Cornell University, Ithaca, New York 14853.
‡ Department of Computer Science, Cornell University, Ithaca, New York 14853.

Matrices of the form (1.6) become ill conditioned as μ tends to zero. Also, when μ is small many of the digits of B will, in general, be lost when H is formed. This loss of information and ill-conditioning can have disastrous effects on methods that rely on having H. There is, as we shall see, a way to solve (1.1) that sidesteps these difficulties.

Our method is an adaptation of the method of Moré and Sorensen [1983] and retains their method's convergence properties. That is, our algorithm will compute a nearly optimal solution in finitely many iterations. More precisely, for $\sigma_1 \in (0, 1)$ the approximate solution s satisfies

$$\psi(s) - \psi(s^*) \leqq \sigma_1(2-\sigma_1)|\psi(s^*)| \quad \text{and} \quad \|s\| \leqq (1+\sigma_1)\Delta \tag{1.7}$$

where s^* is a solution to (1.1).

The paper is organized as follows. The next section briefly reviews the theoretical background needed to develop a suitable algorithm. Readers wishing a more thorough treatment should see Moré and Sorensen [1983]. The algorithm is constructed in § 3. Last, in § 4 we give a detailed report of numerical tests comparing TRSQPF with GQTPAR on trust region problems.

2. Structure of the problem. In this section we will show that the trust region problem usually reduces to a zero finding problem of a rational function with second-order poles. This perspective will enable us to construct an effective algorithm with good convergence properties. The next two lemmas provide necessary and sufficient conditions for a point $s \in R^n$ to be a solution to (1.1).

LEMMA 2.1. *If s is a solution to* (1.1), *then s is a solution to an equation of the form*

$$(H+\lambda I)s = -g \tag{2.2}$$

with $H+\lambda I$ positive semidefinite, $\lambda \geqq 0$, and $\lambda(\Delta - \|s\|) = 0$.

LEMMA 2.3. *Let $s \in R^n$ satisfy* (2.2) *with $H+\lambda I$ positive semidefinite.*

(i) *If $\lambda = 0$ and $\|s\| < \Delta$, then s solves* (1.1);

(ii) *s solves $\psi(s) = \min\{\psi(w)\colon \|w\| = \|s\|\}$*;

(iii) *If $\lambda \geqq 0$ and $\|s\| = \Delta$, then s solves* (1.1).

For proofs of these lemmas see Sorensen [1982]. Moré and Sorensen [1983] also show that (1.1) has no solutions s with $\|s\| = \Delta$ if and only if H is positive definite and $\|H^{-1}g\| < \Delta$. The important consequence of Lemma 2.3 is that when (1.1) has a solution on the boundary of $\{w\colon \|w\| \leqq \Delta\}$, then that solution can be found by solving

$$\|s_\lambda\|^2 - \Delta^2 = 0, \quad \text{where } s_\lambda = -(H+\lambda I)^{-1}g. \tag{2.4}$$

Thus, except for a special case, (1.1) has been reduced to a zero-finding problem in one variable, λ. The next crucial observation, due to Reinsch [1967], [1971] and Hebden [1973], is that (2.4) is a rational function in λ with second-order poles on a subset of the negatives of the eigenvalues of the matrix H. This follows easily from the Real-Schur decomposition:

$$H = Q\Lambda Q^T \quad \text{where } \Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \cdots, \lambda_n) \text{ and } Q^TQ = I. \tag{2.5}$$

Moreover, we assume that the eigenvalues are ordered so that $\lambda_1 \leqq \cdots \leqq \lambda_n$. We now have

$$\|s_\lambda\|^2 = \|Q(\Lambda+\lambda I)^{-1}Q^Tg\|^2 = \sum_{i=1}^{n} \frac{\gamma_i^2}{(\lambda_i+\lambda)^2} \tag{2.6}$$

where γ_i is the ith component of Q^Tg. Let S_1 denote the eigenspace corresponding to the smallest eigenvalue λ_1 of H. If g has a component in S_1, then (2.4) has a solution in $(-\lambda_1, \infty)$, and this is called the "general case." If, however, g is perpendicular to S_1, then (2.4) may not have a solution in $(-\lambda_1, \infty)$, and this leads to numerical difficulties. This situation is called the "hard case." As Moré and Sorensen point out, the characteristic difficulty of the hard case is that $\|s_\lambda\| < \Delta$ whenever $H+\lambda I$ is positive definite. In the hard case, a solution to (1.1) can be obtained by solving

$$(H-\lambda_1 I)s = -g \tag{2.7}$$

for s with $\|s\| < \Delta$ and by determining an eigenvector $z \in S_1$. Then

$$(H-\lambda_1 I)(s+\tau z) = -g \tag{2.8}$$

and $\|s+\tau z\| = \Delta$ for some τ. Lemma 2.3 now shows that $s+\tau z$ solves (1.1).

It is now easier to see how the ill-conditioning in matrices of the form (1.6) causes problems. Specifically, the solution to (2.2) becomes unstable as μ tends to zero. Gould [1986] offers a numerically stable way to solve such systems. Essentially the idea is to avoid using the true Hessian and instead compute the trust region steps with an extended matrix whose condition number is normally much better. If $H+\lambda I = B+(1/\mu)AA^T+\lambda I$, as in (1.6), we then set

$$X = \begin{pmatrix} B+\lambda I & A \\ A^T & -\mu I \end{pmatrix} \tag{2.9}$$

and if

$$X\begin{pmatrix} s_\lambda \\ r \end{pmatrix} = \begin{pmatrix} -\nabla f \\ -c \end{pmatrix} \tag{2.10}$$

for some $s_\lambda \in R^n$ and $r \in R^t$, then $(H+\lambda I)s = -\nabla f(x) - (1/\mu)Ac(x) = -g$. This way of solving (2.2) will play an instrumental role in the algorithm for solving (1.1). It is important to note that X does become ill conditioned as $\mu \to 0$ if A is nearly singular. While there is a way to cope with that situation we shall reserve the discussion for another paper. For now we assume that A has full column rank in a neighborhood of the minimum of $p(x)$. Every iteration of our algorithm will need to know if $H+\lambda I$ is positive definite. Therefore, the following definition and lemma are in order.

DEFINITION 2.11. The inertia of an $n \times n$ symmetric matrix H is the triple In $(H) = (a, b, c)$ where a, b, c are, respectively, the number of positive, negative, and zero eigenvalues of H, respectively.

LEMMA 2.12. *$In(H+\lambda I) = (a, b, c)$ if and only if $In(X) = (a, b+t, c)$.*

Proof. See Gould [1986] for the proof. □

Note that Lemma 2.12 implies that the extended matrix X has at least t negative eigenvalues and so cannot be positive definite. Thus we simply cannot use the Cholesky factorization to solve (2.10). Fortunately, the symmetric indefinite factorization of Bunch and Kaufman [1977] is ideally suited to this situation. The Bunch-Kaufman factorization factors the extended system as $X = US^{-1}SDSS^{-1}U^T$ where U is a product of unit upper triangular and permutation matrices. That is,

$$U = P_{n+t}U_{n+t}^{-1} \cdots P_1 U_1^{-1} \quad \text{and} \quad U_k^{-1} = \begin{pmatrix} I_{k-1} & -m_k & 0 \\ 0 & I_l & 0 \\ 0 & 0 & I_{n+t-k} \end{pmatrix}$$

(with $l = 1$ or 2) and D is a block diagonal matrix with blocks of order one or two, and S is an optional diagonal scaling matrix chosen so that the entries of US^{-1} are bounded. Moreover, it turns out that the inertia of X can be determined easily from

D. This property along with Lemma 2.12 will be very useful to our algorithm. For more details on the Bunch-Kaufman factorization, see the Linpack Users' Guide (Dongarra et al. [1979] and Bunch and Kaufman [1977]).

3. The algorithm. As in § 2 the development here will be much the same as that of Moré and Sorensen [1983] but will include those modifications necessary to make the algorithm perform well on penalty functions. The algorithmic development begins with a search for a good way to solve

$$\Phi(\lambda)=\|s_\lambda\|^2-\Delta^2=0 \quad \text{where } s_\lambda=-(H+\lambda I)^{-1}g \text{ and } \lambda>-\lambda_1(H). \tag{3.1}$$

It is well known that $\Phi(\lambda)$ is convex and decreasing on $(-\lambda_1,\infty)$. When Newton's method is applied to such functions, λ_+ (the updated value of λ) is always less than λ^*, and if $\lambda>\lambda^*$ then λ_+ may be much less than λ^*. Dennis and Schnabel [1983] have attempted to overcome the problem of excessively short or long steps by modeling (3.1) with a single term rational function and have derived the following update:

$$\lambda_+=\lambda+\left(\frac{\|s_\lambda\|^2}{\Delta}\right)\left[\frac{\|s_\lambda\|-\Delta}{s_\lambda^T(H+\lambda I)^{-1}s_\lambda}\right]. \tag{3.2}$$

Moré and Sorensen [1983] obtain the same update by applying Newton's method to the function

$$\phi(\lambda)=\frac{1}{\Delta}-\frac{1}{\|s_\lambda\|}=0. \tag{3.3}$$

It follows from (2.6) that $\phi(\lambda)$ is convex and twice continuously differentiable on the interval $(-\lambda_1,\infty)$. In practice we have observed that (3.1) works well when $-\lambda_1<\lambda<\lambda^*$ but it still generally overcorrects when $\lambda>\lambda^*$. This is not surprising since a one-term approximation to (3.1) is valid only near $-\lambda_1$. The following example typifies the difficulties encountered when trying to compute trust region steps for the L_2 penalty function. Let $\mu=10^{-2}$, $\Delta=1$, $c=1$, and

$$B=\begin{pmatrix}-.5 & 1.5\\ 1.5 & -.5\end{pmatrix},\quad A=\begin{pmatrix}.5\\ 1\end{pmatrix},\quad \text{and hence } H=\begin{pmatrix}24.5 & 51.5\\ 51.5 & 99.5\end{pmatrix}. \tag{3.4}$$

Moreover, if we let $\nabla f=[-3,2]^T$, then $g=[47,102]^T$. Then eigenvalues of H are $\lambda_1=-1.7064$ and $\lambda_2=125.7064$ and the value of λ^* for the trust region problem defined by these quantities is 9.5377. It follows that for $\lambda>21$ equation (3.2) will return a value of $\lambda_+<-\lambda_1=1.7064$. When that happens, λ must be updated in some other way. While the interval $(1.706, 21)$ may appear to provide a relatively large radius of convergence for Newton's method it should also be noted that a commonly used starting value

$$\lambda=\frac{\|g\|}{\Delta}=112.3076 \tag{3.5}$$

is much larger than 21. Therefore, to construct an efficient algorithm for solving trust region problems in this setting we must derive a better starting value for λ and devise a more robust way to update λ.

If a method is to produce a good update for λ when $\lambda>\lambda^*$ it must in some way pay attention to the curvature of ϕ. With this goal in mind we will try to construct a meaningful quadratic model of ϕ and then let λ_+ be the left root of the model. For the same reason Newton's method failed when $\lambda>\lambda^*$, the usual quadratic model (i.e., the first three terms of a Taylor series) also often produces a poor update in this setting.

That is, $\phi''(\lambda)$ tends to zero as λ goes to infinity (actually, $\phi''(\lambda)=O(1/\lambda)$ as $\lambda\to\infty$) so that any quadratic model built on local information at λ will not have enough curvature and will therefore greatly underestimate λ^*. A more successful approach is to construct a quadratic model with information acquired at points left and right of λ^*. As we shall see, when $\lambda>\lambda^*$ the algorithm has on hand a lower bound λ_S for $-\lambda_1(H)$ and an approximate unit eigenvector $\hat{z}$ corresponding to $\lambda_1(H)$. Moreover, $\phi(-\lambda_1)=1/\Delta$ and $\phi'(-\lambda_1^+)=-1/|v_1^Tg|\phi'(-\lambda_1^+)$ denotes the derivative from the right, where v_1 is the exact eigenvector corresponding to λ_1. Thus it seems reasonable to use λ_S and $\hat{z}$ in scheme for updating λ. We are now poised to state the basic algorithm.

ALGORITHM 3.6. Let $\lambda\geqq 0$ with $H+\lambda I$, as in (1.6), positive definite, and $\Delta>0$ be given.

```
(1) Form $X$ and compute its Bunch-Kaufman factorization.
(2) Solve $X\binom{s}{r}=\binom{-\nabla f}{-c}$, as in (2.9).
(3) Solve $X\binom{q}{w}=\binom{s}{0}$.
(4) **If** $\|s\|>\Delta$ **then**
      Update $\lambda$ via (3.2)
    **Elseif** $\phi(\lambda)>-1/\Delta$ **then**
      Determine the quadratic, $\chi(\lambda)$, which interpolates the points $(\lambda_S, 1/\Delta)$,
      $(\lambda, \phi(\lambda))$ and satisfies $\chi'(\lambda)=\phi'(\lambda)$.
      Let $\lambda_{1+}$ be the left root of $\chi(\lambda)$.
      Let $\lambda_{2+}$ be the left root of the best local quadratic approximation to $\phi$ at
      $\lambda$ (i.e., the first three terms of a Taylor series).
      **If** $\lambda_{2+}>\lambda_S$ **then**
        Let $\lambda_+=\min\{\lambda_{1+},\lambda_{2+}\}$.
      **Else**
        $\lambda_+=\lambda_{1+}$
    **Else**
      Determine the quadratic, $\chi(\lambda)$, which interpolates the points $(\lambda_S, 1/\Delta)$,
      $(\lambda, \phi(\lambda))$ and satisfies $\chi'(\lambda_S)=-1/|\hat{z}^Tg|$.
      Let $\lambda_+$ be the left root of $\chi(\lambda)$.
```

Note that where possible we have used the best local model of ϕ. This has been done to ensure quadratic convergence near λ^*. In practice we have seen that Algorithm 3.6 is usually robust enough to converge from almost any reasonable starting value. However, to ensure convergence Algorithm 3.6 must, especially in the hard case, be safeguarded. Also, trust region methods usually require only an approximate solution. Thus, it remains to develop good convergence criteria. We shall postpone these matters until we have dealt effectively with the hard case.

In § 2 we have indicated that an eigenvector corresponding to λ_1 is required to solve the hard case. Moré and Sorensen [1983] show that a good approximate eigenvector can provide an acceptable inexact solution.

LEMMA 3.7. *Let $0<\sigma<1$ be given and suppose that for $\lambda\geqq 0$, $H+\lambda I$ is positive definite and $(H+\lambda I)s_\lambda=-g$. Let $z\in R^n$ satisfy*

$$\|s_\lambda+z\|=\Delta \quad and \quad z^T(H+\lambda I)z\leqq\sigma[s_\lambda^T(H+\lambda I)s_\lambda+\lambda\Delta^2)]; \tag{3.8}$$

then

$$-\psi(s_\lambda+z)\geqq\tfrac{1}{2}(1-\sigma)[s_\lambda^T(H+\lambda I)s_\lambda+\lambda\Delta^2]\geqq(1-\sigma)|\psi(s^*)| \tag{3.9}$$

where $\psi(s^*)$ is the optimal value of (1.1).

Proof. See Moré and Sorensen [1983] for the proof. □

It follows from Lemma 3.7 that $|\psi(s_\lambda+z)-\psi(s^*)|\leqq\sigma|\psi(s^*)|$ and so $s_\lambda+z$ is a nearly optimal solution to (1.1). As Moré and Sorensen suggest, anytime we have $\|s_\lambda\|<\Delta$ we try to satisfy (3.8) with $z=\tau\hat{z}$ where $\|s_\lambda+\tau\hat{z}\|=\Delta$ and $\hat{z}$ is an approximate eigenvector of unit norm corresponding to $-\lambda_1$. Furthermore, our complete algorithm (to be given later) will require that $\hat{z}$ satisfy $\hat{z}^T(H+\lambda I)\hat{z}\to 0$ as $\lambda\to-\lambda_1^+$. Such a vector $\hat{z}$ with $\|\hat{z}\|=1$ can be obtained with the following algorithm.

ALGORITHM 3.10. Given $H+\lambda I$ positive definite, the extended system X, and its $US^{-1}SDSS^{-1}U^T$ factorization, the following algorithm computes a vector $\hat{z}$ such that $\|\hat{z}\|=1$ and $\hat{z}^T(H+\lambda I)\hat{z}$ is as small as possible.

(1) If $U=P_nU_n^{-1}\cdots P_1U_1^{-1}$, let $P=P_1P_2\cdots P_n$ and let $\tilde{U}=PU$. Note that $\tilde{U}$ is upper triangular.
(2) Compute the QR factorization of $\tilde{U}DS$. We hasten to point out that since D is tridiagonal, the product $\tilde{U}DS$ is upper Hessenberg and so its QR factorization can be computed in $O(n+t)$ flops.
(3) Solve $Rw=e$ where e is a vector of the form $(\pm1,\cdots,\pm1)^T$ using the following procedure

```
    $\theta_i=0$ for $i=1,\cdots,n+t$
    For $k=n+t,\cdots,1$
```

$$w_k^+=(1+\theta_k)/r_{kk}$$

$$w_k^-=(-1+\theta_k)/r_{kk}$$

$$\nu^+=|w_k^+|+\sum_{i=1}^{k-1}|\theta_i+r_{ik}w_k^+|$$

$$\nu^-=|w_k^-|+\sum_{i=1}^{k-1}|\theta_i+r_{ik}w_k^-|$$

```
        If $\nu^+\geqq\nu^-$
            then $w_k=w_k^+$
            else $w_k=w_k^-$
        $\theta_i=\theta_i+r_{ik}w_k$ for $i=1,\cdots,k-1$
```

(4) Solve $S^{-1}\tilde{U}^Tx=w$.
(5) Solve $Py=x$.
(6) If $y=(z_1,z_2)^T$ with $z_1\in R^n$ and $z_2\in R^t$ we then set $\hat{z}=z_1/\|z_1\|$.

As we said before the only property of $\hat{z}$ required by the complete algorithm is that $\hat{z}^T(H+\lambda I)\hat{z}$ approach zero as λ approaches $-\lambda_1$. The following lemma confirms this.

LEMMA 3.11. *If* $\lambda\to-\lambda_1^+$, *then for the* $\hat{z}$ *from Algorithm* 3.10, $\hat{z}^T(H+\lambda I)\hat{z}\to 0$.

Proof. See Appendix A for the proof. □

Last, given $\hat{z}$ from Algorithm 3.10 the proof of Lemma 3.7 shows that the correct choice of τ is

$$\tau=\frac{\Delta^2-\|s_\lambda\|^2}{s_\lambda^T\hat{z}+\operatorname{sgn}(s_\lambda^T\hat{z})[(s_\lambda^T\hat{z})^2+(\Delta^2-\|s_\lambda\|^2)]^{1/2}}. \tag{3.12}$$

Algorithm 3.10 will not, in general, be used at every iteration where $\|s_\lambda\|<\Delta$, but rather to generate starting values for inverse iteration. If at some iteration we have $\|s_\lambda\|<\Delta$ and we have an approximate eigenvector $\hat{z}$ of unit norm from a previous iteration (where $\lambda_p\geqq-\lambda_1$),we then solve $(H+\lambda I)y=\hat{z}$ and if

$$\hat{y}^T(H+\lambda I)\hat{y}<\varepsilon\hat{z}^T(H+\lambda_pI)\hat{z}\quad\text{where } \hat{y}=y/\|y\|$$

and $\varepsilon \in (0,1)$ is some fixed constant, we then take $\hat{y}$ to be our new approximate eigenvector. Otherwise we resort to Algorithm 3.10. In practice Algorithm 3.10 is seldom used more than twice within each instance of (1.1).

As it stands it is possible, although unlikely, for Algorithm 3.6 to return a value of λ_+ that is less than $-\lambda_1$. To prevent this and to guarantee that a solution will be found in finitely many iterations, the algorithm must be safeguaraded. Moré and Sorensen's safeguarding scheme is perfectly adequate for this setting and so we use it. An interval containing λ^*, $[\lambda_L, \lambda_U]$, and a lower bound λ_S, for $-\lambda_1$ are maintained at all times. At the beginning of each iteration the value of λ_+ inherited from the previous iteration is checked against these parameters by the following procedure.

```
Safeguard $\lambda$:
    (1) $\lambda = \max(\lambda, \lambda_L)$,
    (2) $\lambda = \min(\lambda, \lambda_U)$,
    (3) If $\lambda \leqq \lambda_S$ then $\lambda = \max\{.001\lambda_U, (\lambda_L, \lambda_U)^{1/2}\}$.
```

As Moré and Sorensen point out, the third step is probably the most important. Without it, we could not prove that the algorithm will yield an approximate solution in finitely many iterations. The procedure for updating λ_L, λ_U, and λ_S is straightforward (for details see Moré and Sorensen [1983]).

```
Update $\lambda_L$, $\lambda_U$, and $\lambda_S$:
    (1) If $H+\lambda I$ is nonsingular then
            Compute $\hat{z}$ as described above
            If $H+\lambda I$ is positive definite then
                $\lambda_S = \max\{\lambda_S, \lambda - \hat{z}^T(H+\lambda I)\hat{z}\}$
            Else
                $\lambda_S = \max\{\lambda_S, \lambda, \lambda - \hat{z}^T(H+\lambda I)\hat{z}\}$
    (2) If $\lambda \in (-\lambda_1, \infty)$ and $\phi(\lambda) < 0$ then
            $\lambda_U = \min(\lambda_U, \lambda)$
        Else
            $\lambda_L = \max(\lambda_L, \lambda)$
    (3) Let $\lambda_L = \max(\lambda_L, \lambda_S)$.
```

Initial values for the safeguarding parameters are:

$$\lambda_S = \max\{-h_{ii}, \text{for } i=1, \cdots, n, -\|B\|_1 \text{ (if } t<n)\},$$

$$\lambda_L = \max\left\{0, \lambda_S, \frac{\|g\|}{\Delta} - \|B\|_1 - \frac{1}{\mu}\|A\|_1\|A\|_\infty\right\},$$

$$\lambda_U = \min\left\{\frac{\|g\|}{\Delta} + \|B\|_1 + \frac{1}{\mu}\|A\|_1\|A\|_\infty, \frac{\|g\|}{\Delta} + \|B\|_1 \text{ (if } t<n)\right\}$$

where h_{ii} is the ith diagonal element of $H = B + (1/\mu)AA^T$.

Since each iteration of the algorithm will require $O(n+t)^3$, it is imperative that we try to minimize the number of iterations needed to find a solution. As we saw earlier in the example, a poor initial value for λ will certainly hinder progress toward a solution. It has been our experience that it is generally worthwhile expending some effort to compute a better starting value for λ. In other words, we were unable to find a starting value for λ that was both inexpensive and guaranteed to be close to the solution. The initial value described below proved itself computationally cost effective when compared to less expensive initial values. To begin the derivation we observe that (2.5) implies

$$\|s_\lambda\|^2 \leqq \sum_{i=1}^{n-t} \frac{\gamma_i^2}{(\lambda_1+\lambda)^2} + \sum_{i=n-t+1}^{n} \frac{\gamma_i^2}{\lambda_{n-t+1}^2} \tag{3.13}$$

and if reasonable estimates for $\sum_{i=1}^{n-t} \gamma_i^2$, $\sum_{i=n-t+1}^{n} \gamma_i^2$, λ_1, and λ_{n-t+1} can be found, then we may be able to use (3.13) to compute a suitable initial value for λ. The estimates should, if possible, improve as μ tends to zero since the accuracy of the initial values becomes increasingly critical as μ becomes small. To secure such estimates we let $A=QR$ where Q is orthogonal and R is upper-triangular. Note that since Q is orthogonal the last $n-t$ columns $[q_{t+1}, \cdots, q_n]$ form an orthonormal basis for the null space of A^T (computing a QR factorization of A is even more useful when A is nearly rank deficient in a neighborhood of the solution but this is the topic of another paper). We now make the following approximations:

$$\sum_{i=1}^{n-t} \gamma_i^2 \approx \sum_{j=t+1}^{n} (q_j^T g)^2 \equiv \delta$$

and

$$\sum_{i=n-t+1}^{n} \gamma_i^2 \approx \|g\|^2 - \delta \equiv \zeta.$$

Unfortunately, we could not find estimates for λ_1 and λ_{n-t+1} that improve as μ tends to zero. However, it is easy to see that if $\lambda_1(H)<0$, then $\lambda_1(B)<\lambda_1(H)$. To compute $\lambda_1(B)$ we reduce B to tridiagonal form with Householder transformations. The characteristic polynomial is then easily formed, and its smallest root can be readily found via bisection. For details see Golub and Van Loan [1983]. If A has full rank it can be shown that as μ tends to zero, $\lambda_{n-t+1}(H)$ converges to $(1/\mu)\lambda_{n-t+1}(AA^T)$. Furthermore,

$$\lambda_{n-t+1}(AA^T) \leqq \frac{1}{\mu} \min \{\bar{r}_{ii}^2, \bar{a}_{ii}\} \equiv \rho \tag{3.14}$$

where $\bar{r}_{ii}$ is the element of smallest absolute value along the diagonal of R, and $\bar{a}_{ii}$ is the smallest diagonal entry of AA^T. And so we use (3.14) as our approximation for λ_{n-t+1}. Substituting δ, ζ, $\lambda_1(B)$, and ρ for their respective quantities in (3.13) and solving for λ gives

$$\lambda \approx \left[\frac{\delta^2}{\Delta^2 - \zeta^2/\rho^2}\right]^{1/2} + |\lambda_1(B)|. \tag{3.15}$$

It is certainly possible for the denominator in (3.15) to be negative and in that case we recommend using $\|B\|_1$ or $\|g\|/\Delta$ for a starting value. Formula (3.15) is most likely to fail when Δ is small or when ρ is small, but in these situations λ^* (the value of λ corresponding to s^*) is generally larger. In other words, λ^* is usually inversely proportional to Δ and ρ. The final ingredient of the iteration is the convergence criteria. Here again we invoke, without modification, Moré and Sorensen's work. For the convergence parameter $\sigma_1 \in (0, 1)$, and a $\lambda \geqq 0$ such that $H+\lambda I$ is positive definite, the candidate step s_λ, computed by Algorithm 3.6, is checked for near optimality with the following scheme.

Convergence Criteria:

```
(1) If $|\Delta-\|s_\lambda\||\leqq\sigma_1\Delta$ or $\|s_\lambda\|\leqq\Delta$, $\lambda=0$ then
        Terminate with $s=s_\lambda$ as the approximate solution.
(2) If $\|s_\lambda\|<\Delta$ then
        Use Algorithm 3.10 to compute $\hat{z}$ and $\tau$
        If $s_\lambda$, $\hat{z}$, and $\tau$ satisfy (3.8) then
            Terminate with $s=s_\lambda+\tau\hat{z}$ as the approximate solution.
```

Moré and Sorensen [1983] show that these criteria guarantee that if the algorithm terminates, then the approximate solution s satisfies

$$\psi(s)-\psi(s^*)\leqq\sigma_1(2-\sigma_1)|\psi(s^*)| \quad \text{and} \quad \|s\|\leqq(1+\sigma_1)\Delta,$$

and thus s is a nearly optimal solution of (1.1). It should be noted that (3.8) provides termination criteria that is applicable to both the general and the hard cases. In our tests we observe that when a very inexact solution (i.e., $\sigma_1=.02$) is sought, termination usually occurs on (3.8) even in the general case. We have now discussed all the ingredients of the iterative scheme for solving problem (1.1). The following algorithm summarizes these ingredients and defines a typical iteration.

ALGORITHM 3.16.

```
(1) Safeguard $\lambda$.
(2) Form $X$ and compute its $US^{-1}SDSS^{-1}U^T$ factorization. If $H+\lambda I$ is not
    positive definite then go to (5).
(3) Solve $(H+\lambda I)s_\lambda=-g$ as in Algorithm 3.6.
(4) If $\|s_\lambda\|<\Delta$ use Algorithm 3.10 to compute $\tau$ and $\hat{z}$.
(5) Update $\lambda_L$, $\lambda_U$, and $\lambda_S$.
(6) Check the convergence criteria.
(7) If $H+\lambda I$ is positive definite and $g\neq0$ then update $\lambda$ via Algorithm 3.6,
    otherwise set $\lambda=\lambda_S$.
```

Last, we must now show that the above algorithm will produce a nearly optimal solution in a finite number of steps.

THEOREM 3.17. *For $\phi(\lambda)$ as in* (3.3) *and any $\varepsilon>0$ Algorithm* 3.16 *will, in finitely many iteration, return a value of λ in $[-\lambda_1,\infty)$ such that one of the following hold*:

(i) $\lambda\in(\lambda^*-\varepsilon,\lambda^*+\varepsilon)$, *where $\lambda^*\in(-\lambda_1,\infty)$ and $\phi(\lambda^*)=0$, in the event of the general case.*

(ii) $\lambda\in[-\lambda_1,-\lambda_1+\varepsilon)$ *in the event of the hard case.*

(iii) $\lambda\in[0,\varepsilon)$ *in the event of the positive definite case.*

Proof. (i) General Case. The proof is by contradiction. Suppose that the length of the interval of uncertainty is bounded away from zero. Then the bisection step of the safeguarding scheme can only be invoked finitely many times. If during some iteration a value of λ is found such that $\phi(\lambda)>0$ and $\lambda>-\lambda_1$, then the convexity of ϕ implies that Newton's method will proceed monotonically to λ^* and a nearly optimal solution will be found in finitely many more iterations.

Now suppose that the length of the interval of uncertainty does not go to zero and the algorithm never finds a value of λ for which $\phi(\lambda)>0$ and $\lambda>-\lambda_1$. This implies that after a finite number of iterations we have a sequence $\{\lambda_k\}_N^\infty$ that satisfies $\lambda_k>\lambda_{k+1}>\lambda^*>-\lambda_1$ and $\phi(\lambda_k)<0$. To show that the sequence $\{\lambda_k\}_N^\infty$ converges, we will compare our method to one that we know converges: the secant method. In particular, we will show that $\lambda_k-\lambda_{k+1}$ is larger for our method than for a suitable and obviously convergent secant method. This result, under all of the assumptions we have already made, will prove the convergence.

Suppose that the exact value of $-\lambda_1$ is known. Then since $\phi(-\lambda_1)=1/\Delta$ we could update λ_k by the secant method applied to the points $(-\lambda_1, 1/\Delta)$ and $(\lambda_k, \phi(\lambda_k))$ and thereby obtain

$$\lambda_a^+=\frac{\lambda_k+\lambda_1}{1-\Delta\phi(\lambda_k)}-\lambda_1.$$

Since ϕ is convex on $(-\lambda_1, \infty)$, it is easy to see that updating λ_k by this formula will generate a sequence converging to λ^*. Now consider the update

$$\lambda_b^+=\frac{\lambda_k+\lambda_S}{1-\Delta\phi(\lambda_k)}+\lambda_S,$$

which was obtained by applying the secant formula to the points $(\lambda_S, 1/\Delta)$, $(\lambda_k, \phi(\lambda_k))$. Since $\lambda_S \leqq -\lambda_1$ it easily follows that $\lambda_b^+ < \lambda_a^+$. Let χ be any convex quadratic that interpolates the points $(\lambda_S, 1/\Delta)$, $(\lambda_k, \phi(\lambda_k))$. Since $1/\Delta > 0$ and $\phi(\lambda_k)<0$, it easily follows that the leftmost root r_l satisfies $\lambda_S < r_l < \lambda_b^+ < \lambda_a^+$. If, on the other hand, χ is concave, then r_l clearly satisfies $r_l < \lambda_S < \lambda_b^+ < \lambda_a^+$. In either case we have $r_l < \lambda_b^+ < \lambda_a^+$, and taking $\lambda_{k+1}=r_l$ it immediately follows that $\lambda_k-\lambda_{k+1} > \lambda_k - \lambda_a^+$, as required. This proves convergence for the general case.

(ii) Hard Case. In this case we have $\phi(-\lambda_1)\leqq 0$ and again we assume that the interval of uncertainty remains bounded away from zero and so after finitely many iterations, we have a sequence $\{\lambda_k\}_N^\infty$ that satisfies $\lambda_k > \lambda_{k+1} > -\lambda_1$ and $\phi(\lambda_k)<0$. As in the general case we will show that our method converges by comparing it to an obviously convergent variant of the secant method. Suppose that the exact values of $-\lambda_1$ and $\phi(-\lambda_1+\delta)$, where $\delta \in (0, \varepsilon)$, are known. Updating λ_k by applying the secant method to the points $(-\lambda_1, 1/\Delta)$ and $(\lambda_k, \phi(-\lambda_1+\delta))$ yields

$$\lambda_a^+=\frac{\lambda_k+\lambda_1}{1-\Delta\phi(-\lambda_1+\delta)}-\lambda_1.$$

Even though the points $(-\lambda_1, 1/\Delta)$ and $(\lambda_k, \phi(-\lambda_1+\delta))$ do not lie on the function ϕ, it is easy to see that the above formula will generate a sequence that converges monotonically to $-\lambda_1$. Consider the following update:

$$\lambda_b^+=\frac{\lambda_k+\lambda_S}{1-\Delta\phi(\lambda_k)}-\lambda_S. \tag{3.18}$$

Since $\lambda_S \leqq -\lambda_1$ and $\phi(\lambda_k) < \phi(-\lambda_1+\delta)$, for $\lambda_k > -\lambda_1+\delta$, we clearly have $\lambda_b^+ < \lambda_a^+$. Thus for any sequence $\{\lambda_k\}_N^\infty$ obtained from updating λ_k by (3.18) it follows that for some finite $K \geqq N$ we have $\lambda_K \leqq -\lambda_1+\delta$. But since δ was arbitrary, the sequence $\{\lambda_k\}_N^\infty$ generated by (3.18) converges to $-\lambda_1$.

(iii) Positive-Definite Case. The proof for this case is identical to the proof of the hard case except that $-\lambda_1$ should be replaced with zero. □

4. Computational results. In this section we report the results of some tests in which we compare the performance of TRSQPF (our algorithm) with that of GQTPAR (Moré and Sorensen's algorithm) on individual trust region problems where g and H are of the form (1.3) and (1.6), respectively. The tests covered a wide range of values of n, t, and μ and for each triple (n, t, μ) Tables 1-12 below summarize the outcome of 10 different trust region problems. Individual problems were created by specifying an $n \times n$ diagonal matrix for D_B, an $n \times t$ diagonal matrix for D_A, an n vector for g, and a t vector for c. By appropriately choosing the entries in each of these quantities we can construct any of the four main types of trust region problems: general case, hard case, positive definite case, and the saddle-point case.

TABLE 1

			General case							
			GQTPAR				TRSQPF			
n	t	μ	# crash	min	max	mean	min	max	mean	adjst. mean
20	5	10^{-2}	0	5	9	6.4	1	5	2.9	3.8
		10^{-5}	0	5	10	7.4	2	4	2.7	3.6
		10^{-9}	0	4	11	8.8	2	4	2.8	3.7
		10^{-12}	0	4	13	9.0	2	4	2.7	3.6
		10^{-16}	8	5	6	5.5	2	5	3.4	4.3
	10	10^{-2}	0	3	9	5.7	1	7	3.9	4.5
		10^{-5}	0	4	10	7.2	1	4	2.5	3.1
		10^{-9}	0	5	14	9.2	2	4	2.9	3.5
		10^{-12}	0	3	11	6.7	2	5	3.0	3.6
		10^{-16}	9	8	8	8.0	2	4	3.1	3.7
	15	10^{-2}	0	3	8	5.3	1	5	3.4	4.0
		10^{-5}	0	6	11	7.4	1	7	3.8	4.4
		10^{-9}	0	5	13	9.4	2	4	2.6	3.2
		10^{-12}	0	2	9	6.1	1	4	2.7	3.3
		10^{-16}	9	7	7	7.0	1	4	3.0	3.6

TABLE 2

			General case							
			GQTPAR				TRSQPF			
n	t	μ	# crash	min	max	mean	min	max	mean	adjst. mean
40	10	10^{-2}	0	3	9	6.7	1	5	3.3	4.5
		10^{-5}	0	5	10	7.3	1	5	3.2	4.4
		10^{-9}	0	5	10	7.9	2	4	3.2	4.4
		10^{-12}	0	3	12	7.6	2	4	3.3	4.5
		10^{-16}	7	2	9	6.0	2	4	3.2	4.4
	20	10^{-2}	0	3	7	5.6	1	6	3.3	4.2
		10^{-5}	0	5	9	7.2	2	6	3.3	4.2
		10^{-9}	0	5	12	8.9	2	4	2.8	3.7
		10^{-12}	0	3	12	6.9	2	4	3.6	4.5
		10^{-16}	7	7	10	8.7	2	5	2.6	3.5
	30	10^{-2}	0	4	9	5.2	2	5	3.2	3.6
		10^{-5}	0	4	10	7.5	1	6	4.2	4.6
		10^{-9}	0	6	13	9.4	2	4	2.8	3.2
		10^{-12}	0	3	12	8.5	2	4	3.2	3.6
		10^{-6}	10	–	–	–	2	5	2.8	3.2

TABLE 3

			General case							
			GQTPAR				TRSQPF			
n	t	μ	# crash	min	max	mean	min	max	mean	adjst. mean
80	20	10^{-2}	0	4	9	6.4	1	6	3.9	5.5
		10^{-5}	0	5	10	7.0	1	5	3.7	5.3
		10^{-9}	0	4	12	7.6	2	4	3.3	4.9
		10^{-12}	0	3	10	7.6	2	5	3.6	5.2
		10^{-16}	5	8	13	6.3	2	5	3.8	5.4
	40	10^{-2}	0	3	9	5.5	1	5	3.5	4.7
		10^{-5}	0	4	11	7.5	4	7	5.3	6.5
		10^{-9}	0	5	13	9.0	3	4	3.3	4.5
		10^{-12}	0	2	12	7.0	2	5	3.5	4.7
		10^{-16}	6	5	8	6.8	2	5	3.2	4.4
	60	10^{-2}	0	3	6	4.6	1	6	3.6	4.5
		10^{-5}	0	3	10	8.1	3	6	5.2	6.1
		10^{-9}	0	5	11	8.7	2	6	3.2	4.1
		10^{-12}	0	3	11	7.5	2	6	3.2	4.1
		10^{-16}	7	2	7	5.3	2	4	2.5	3.4

TABLE 4

			Hard case							
			GQTPAR				TRSQPF			
n	t	μ	# crash	min	max	mean	min	max	mean	adjst. mean
20	5	10^{-2}	0	4	6	5.4	2	4	2.8	3.8
		10^{-5}	0	4	11	7.8	3	5	3.3	4.2
		10^{-9}	0	5	11	8.9	2	6	3.5	4.4
		10^{-12}	0	3	12	8.1	2	5	3.5	4.4
		10^{-16}	9	2	2	2.0	2	5	3.6	4.5
	10	10^{-2}	0	3	9	5.9	2	5	3.9	4.5
		10^{-5}	0	4	11	7.5	2	4	2.9	3.5
		10^{-9}	0	5	13	9.3	2	5	3.3	3.9
		10^{-12}	0	3	12	7.8	3	6	4.2	4.8
		10^{-16}	8	6	7	6.5	3	5	3.8	4.4
	15	10^{-2}	0	3	8	5.7	1	4	2.3	2.9
		10^{-5}	0	4	10	7.7	1	6	3.8	4.4
		10^{-9}	0	6	10	8.5	2	7	4.6	5.2
		10^{-12}	0	4	12	8.9	2	6	4.4	5.0
		10^{-16}	9	7	7	7.0	2	7	4.1	4.7

TABLE 5

			Hard case							
			GQTPAR				TRSQPF			
n	t	μ	# crash	min	max	mean	min	max	mean	adjst. mean
40	10	10^{-2}	0	3	9	5.5	1	5	3.6	4.8
		10^{-5}	0	5	10	7.3	1	4	3.0	4.2
		10^{-9}	0	4	12	8.1	2	4	3.5	4.7
		10^{-12}	0	2	12	7.3	1	6	3.3	4.5
		10^{-16}	6	7	11	8.8	2	5	3.7	4.9
	20	10^{-2}	0	1	8	5.0	1	7	3.3	4.2
		10^{-5}	0	4	9	6.5	2	6	3.9	4.8
		10^{-9}	0	5	14	8.8	2	4	3.0	3.9
		10^{-12}	0	2	13	8.0	2	5	3.5	4.4
		10^{-16}	7	9	11	9.7	3	5	3.7	4.6
	30	10^{-2}	0	3	8	5.4	1	6	3.6	4.0
		10^{-5}	0	4	10	7.5	2	7	5.0	5.4
		10^{-9}	0	6	12	9.4	3	6	3.9	4.3
		10^{-12}	0	3	11	7.9	3	7	4.0	4.4
		10^{-16}	8	5	11	8.0	2	6	4.5	4.9

TABLE 6

			Hard case							
			GQTPAR				TRSQPF			
n	t	μ	# crash	min	max	mean	min	max	mean	adjst. mean
80	20	10^{-2}	0	3	7	5.1	1	6	3.5	5.1
		10^{-5}	0	4	11	7.5	2	5	3.3	4.9
		10^{-9}	0	5	12	8.5	2	4	3.5	5.1
		10^{-12}	0	3	11	7.3	2	4	3.4	5.0
		10^{-16}	5	2	9	8.8	2	4	3.0	4.6
	40	10^{-2}	0	3	7	5.5	1	4	3.7	4.9
		10^{-5}	0	4	11	7.2	2	7	4.4	5.6
		10^{-9}	0	5	13	9.4	2	4	3.3	4.5
		10^{-12}	0	3	11	7.3	2	4	3.4	4.6
		10^{-16}	8	4	8	6.0	2	4	3.3	4.5
	60	10^{-2}	0	3	7	4.4	1	6	2.9	3.8
		10^{-5}	0	4	10	7.6	2	7	4.8	5.7
		10^{-9}	0	5	11	8.2	2	5	3.7	4.6
		10^{-12}	0	3	11	8.3	2	4	3.3	4.2
		10^{-16}	3	6	9	7.8	2	5	3.5	4.4

TABLE 7

			Positive-definite case							
			GQTPAR				TRSQPF			
n	t	μ	# crash	min	max	mean	min	max	mean	adjst. mean
20	5	10^{-2}	0	2	5	4.0	1	3	2.0	2.9
		10^{-5}	0	4	5	4.3	1	2	1.9	2.8
		10^{-9}	0	3	5	4.2	1	4	2.2	3.1
		10^{-12}	0	2	4	3.4	2	3	2.5	3.4
		10^{-16}	10	–	–	–	2	4	2.4	3.3
	10	10^{-2}	0	2	7	4.3	2	5	3.5	4.1
		10^{-5}	0	2	5	4.2	2	4	2.4	3.0
		10^{-9}	0	4	5	4.3	2	5	2.5	3.1
		10^{-12}	0	2	4	3.0	1	3	2.9	3.5
		10^{-16}	8	7	17	12.0	2	3	2.2	2.8
	15	10^{-2}	0	2	6	4.0	2	5	2.8	3.4
		10^{-5}	0	2	5	3.4	1	4	2.6	3.2
		10^{-9}	0	2	4	3.1	2	2	2.0	2.6
		10^{-12}	0	2	5	2.7	2	3	2.1	2.7
		10^{-16}	8	6	6	6.0	2	3	2.2	2.8

TABLE 8

			Positive-definite case							
			GQTPAR				TRSQPF			
n	t	μ	# crash	min	max	mean	min	max	mean	adjst. mean
40	10	10^{-2}	0	4	5	4.9	1	5	3.5	4.7
		10^{-5}	0	4	5	4.4	2	3	2.1	3.3
		10^{-9}	0	4	5	4.8	2	3	2.4	3.6
		10^{-12}	0	2	5	3.6	2	4	2.3	3.5
		10^{-16}	4	2	11	6.5	1	2	1.9	3.1
	20	10^{-2}	0	3	6	4.7	1	5	3.1	4.0
		10^{-5}	0	4	5	4.8	2	3	2.4	3.3
		10^{-9}	0	4	5	4.6	1	3	2.0	2.9
		10^{-12}	0	2	5	3.8	2	6	2.6	3.5
		10^{-16}	7	2	7	4.0	1	3	2.2	3.1
	30	10^{-2}	0	2	6	3.9	1	3	2.1	2.5
		10^{-5}	0	3	5	3.9	2	4	3.3	3.7
		10^{-9}	0	2	5	3.5	2	5	2.4	2.8
		10^{-12}	0	2	4	3.4	2	3	2.4	2.8
		10^{-16}	10	–	–	–	2	3	2.4	2.8

TABLE 9

			Positive-definite case							
			GQTPAR				TRSQPF			
n	t	μ	# crash	min	max	mean	min	max	mean	adjst. mean
80	20	10^{-2}	0	3	6	5.0	1	5	2.8	4.4
		10^{-5}	0	5	5	5.0	2	3	2.1	3.7
		10^{-9}	0	4	5	4.9	2	2	2.0	3.6
		10^{-12}	0	2	5	3.5	1	3	2.0	3.6
		10^{-16}	5	3	9	7.8	2	3	2.2	3.8
	40	10^{-2}	0	2	6	4.9	1	5	3.7	4.9
		10^{-5}	0	4	5	4.7	1	5	3.2	4.4
		10^{-9}	0	4	5	4.7	2	3	2.1	3.3
		10^{-12}	0	2	5	3.4	1	3	2.1	3.3
		10^{-16}	9	5	5	5.0	2	3	2.4	3.6
	60	10^{-2}	0	2	8	4.2	1	5	3.1	4.0
		10^{-5}	0	2	6	4.9	2	5	3.7	4.6
		10^{-9}	0	2	5	4.2	2	4	2.3	3.2
		10^{-12}	0	2	5	3.5	1	3	2.1	3.0
		10^{-16}	6	8	11	9.5	2	3	2.1	3.0

TABLE 10

			Saddle-point case							
			GQTPAR				TRSQPF			
n	t	μ	# crash	min	max	mean	min	max	mean	adjst. mean
20	5	10^{-2}	0	4	12	7.4	1	4	1.3	2.1
		10^{-5}	0	8	17	10.2	1	5	1.7	2
		10^{-9}	0	9	18	12.4	1	3	1.4	2.3
		10^{-12}	0	8	19	13.9	1	2	1.1	2.0
		10^{-16}	10	–	–	–	1	4	1.9	2.8
	10	10^{-2}	0	8	14	10.6	1	6	2.9	3.5
		10^{-5}	0	6	13	9.7	1	11	2.5	3.1
		10^{-9}	0	10	17	13.3	1	7	2.1	2.7
		10^{-12}	1	11	16	13.1	1	7	2.7	3.3
		10^{-16}	9	12	12	12.0	1	6	2.8	3.4
	15	10^{-2}	0	6	12	9.7	1	8	5.3	5.9
		10^{-5}	0	2	15	11.8	3	10	5.4	6.0
		10^{-9}	0	2	17	11.0	1	6	4.1	4.7
		10^{-12}	0	12	18	15.3	1	6	3.7	4.3
		10^{-16}	10	–	–	–	1	6	3.4	4.0

TABLE 11

			Saddle-point case							
			GQTPAR				TRSQPF			
n	t	μ	# crash	min	max	mean	min	max	mean	adjst. mean
40	10	10^{-2}	0	6	16	9.5	1	2	1.3	2.5
		10^{-5}	0	6	15	11.0	1	2	1.1	2.3
		10^{-9}	0	8	15	12.2	1	5	1.4	2.6
		10^{-12}	0	9	17	11.4	1	4	1.3	2.5
		10^{-16}	9	13	13	13.0	1	5	2.3	3.5
	20	10^{-2}	0	7	10	7.8	1	5	1.9	2.8
		10^{-5}	0	7	19	11.1	1	4	2.0	2.9
		10^{-9}	0	7	18	13.1	1	6	2.1	3.0
		10^{-12}	0	10	16	13.4	1	5	2.5	3.4
		10^{-16}	9	14	14	14.0	1	9	2.8	3.7
	30	10^{-2}	0	7	15	11.5	1	6	4.5	4.9
		10^{-5}	0	7	14	11.4	1	6	4.1	4.5
		10^{-9}	0	10	15	12.8	1	7	3.6	4.0
		10^{-12}	0	8	16	13.3	1	6	3.7	4.1
		10^{-16}	9	16	16	16.0	1	6	3.0	3.4

TABLE 12

			Saddle-point case							
			GQTPAR				TRSQPF			
n	t	μ	# crash	min	max	mean	min	max	mean	adjst. mean
80	20	10^{-2}	0	7	14	10.1	1	3	1.4	3.0
		10^{-5}	0	8	13	10.9	1	3	1.5	3.1
		10^{-9}	0	8	14	11.6	1	4	1.6	3.2
		10^{-12}	0	10	16	13.5	1	2	1.1	2.7
		10^{-16}	8	10	12	11.0	1	1	1.0	2.6
	40	10^{-2}	0	9	16	11.2	1	4	2.1	3.3
		10^{-5}	0	7	15	11.9	1	4	1.8	3.0
		10^{-9}	0	11	17	13.3	1	3	1.5	2.7
		10^{-12}	0	9	18	13.3	1	4	1.6	2.8
		10^{-16}	8	12	16	13.3	1	5	1.9	3.1
	60	10^{-2}	0	6	13	9.1	1	5	3.2	4.1
		10^{-5}	0	6	15	10.2	1	5	2.9	3.8
		10^{-9}	0	11	18	13.7	1	6	3.4	4.3
		10^{-12}	0	11	18	14.9	1	7	3.0	3.9
		10^{-16}	10	–	–	–	1	5	2.4	3.3

An instance of the general case is formed by choosing the entries of D_B and D_A as uniformly distributed random numbers in $(-1, 1)\backslash 0$ and $(-1, -.2) \cup (.2, 1)$, respectively. The entries of D_A were so chosen to avoid the case where A is nearly rank deficient. The entries of g were chosen as uniformly distributed random numbers in $(-1, 1)$. However, the entries of the constraint vector c were restricted to the interval $(-\mu_{\text{prev}}, \mu_{\text{prev}})$ where μ_{prev} is the next largest value of μ used in the testing. For example, if the current value of μ is 10^{-12}, then μ_{prev} is 10^{-9}. We feel this scaling of c provides a more realistic test because when TRSQPF is used in the context equality constrained minimization, the magnitudes of the constraints are, at any given iteration, usually of the order of the previous value of μ.

To form an instance of the hard case we set to zero the component of g corresponding to the smallest element of $\{D_{B t+1,t+1}, \cdots, D_{Bnn}\}$. If none of those entries of D_B are negative, then $D_{Bn,n}$ is chosen from the interval $(-1, 0)$, and g_n is set to zero. We need only consider the last $n-t$ elements of D_B because for the values of μ that we use $\{10^{-2}, 10^{-5}, 10^{-9}, 10^{-12}, 10^{-16}\}$ and our choice for the entries of D_A ensure that the first t eigenvalues $D_B + (1/\mu)D_A D_A^T$ are positive. An instance of the positive-definite case was formed by choosing $[D_{Bt+1,t+1}, \cdots, D_{Bn,n}]$ from the interval $(0, 1)$. The saddle-point case is somewhat more involved and will be described a little later.

The next step in constructing a trust region problem is to scramble the structure of D_B and D_A. We set $B = QD_BQ^T$ and $A = QD_AZ$ where Q and Z are, respectively, $n \times n$ and $t \times t$ orthogonal matrices. Both Q and Z are of the form $Q_1Q_2Q_3$ or $Z_1Z_2Z_3$ where

$$Q_i = I - 2\frac{v_i v_i^T}{v_i^T v_i} \quad \text{and} \quad Z_i = I - 2\frac{w_i w_i^T}{w_i^T w_i}, \quad i = 1, 2, 3. \tag{4.1}$$

The entries of v_i and w_i are random numbers uniformly distributed in $(-1, 1)$. We are now better able to describe the saddle-point case. Quite simply, an instance of the saddle-point case can be created by setting $g = -(1/\mu)Ac$. Last, the values of Δ were random numbers chosen from the interval $(0, 10)$.

For the purposes of the testing we did not use the initial values for the safeguarding parameters described in § 3. Instead we used the following cruder bounds:

$$\lambda_S = \max\{-h_{ii}, i = 1, \cdots, n,\},$$

$$\lambda_B = \max\left\{0, \lambda_S, \frac{\|g\|}{\Delta} - \|B\|_1 - \frac{1}{\mu}\|A\|_1\|A\|_\infty\right\},$$

$$\lambda_U = \frac{\|g\|}{\Delta} + \|B\|_1 + \frac{1}{\mu}\|A\|_1\|A\|_\infty.$$

These initial values were used to make for a more stringent test of the rest of the algorithm.

The charts also deserve some explanation. For both GQTPAR and TRSQPF we report the minimum, maximum, and average number of iterations used to solve 10 distinct trust region problems for each triple (n, t, μ). For GQTPAR we also note the number of problems (out of 10) on which it ran for 20 iterations. Moré and Sorensen interpret this as a failure to solve the problem and so we report it under the #crash column. The subsequent entries in the min, max, and mean columns were computed from the results of those problems on which GQTPAR halted in less than 20 iterations. Furthermore, to account for the work done by TRSQPF to compute the initial value for α, we add an appropriate amount to the average number of iterations. For specific values of n and t, the number of floating point operations (flops) required to compute the initial α is

$$\frac{2}{3}n^3+t^2\left(n-\frac{t}{3}\right) \tag{4.2}$$

while the work count per iteration of TRSQPF is given by the formula

$$\frac{(n+t)^3}{6}+7(n+t)^2. \tag{4.3}$$

The adjustment is made by evaluating the above formulas and dividing (4.3) into (4.2). For example, for $n=40$ and $t=20$, the ratio of (4.2) to (4.3) is .9 and this is the amount we add to the average number of iterations. Thus, the average number flops per problem can be obtained by multiplying the adjusted mean number of iterations times the number of operations per iteration (i.e., equation (4.3)). In all problems $\sigma_1=0.01$ and $\sigma_2=0.0$.

To conclude this section we would like to offer explanations for some of the results presented in the Tables 1–12. When comparing average numbers of iterations we can see that TRSQPF compares quite favorably with GQTPAR in the general and hard cases. On the basis of some informal experimentation we believe that both the new starting value for λ (i.e., equation (3.21)) and the more complicated way of updating λ (i.e., Algorithm 3.6) are responsible. In the hard case TRSQPF is aided further by the condition estimator put forth in Algorithm 3.10. By contrast, the average numbers of iterations in the positive-definite case are nearly the same except when $\mu=10^{-16}$. In the positive-definite case both algorithms proceed until a candidate value of λ is found that is greater than $-\lambda_1$ and less than zero. When that happens, the safeguarding scheme resets λ to zero, and the problem is solved. Since neither algorithm is especially well tailored to the positive-definite case, neither is especially adept at it, and so the results are unsurprisingly similar. The saddle-point case is where TRSQPF registers the sharpest improvement over GQTPAR. The saddle-point case is, essentially, a search for the direction of most negative curvature. It stands to reason then that the new initial value for λ and the condition estimator in Algorithm 3.10 are the primary factors contributing to TRSQPF's good performance in the saddle-point case.

In all of the testing presented here, we set $\sigma_1=0.01$ and $\sigma_2=0.0$ that essentially require both algorithms to return a solution in which the first two significant digits of the trust region step are correct. We feel this represents a realistic test because within the context of a global algorithm to minimize penalty functions and ultimately solve equality constrained minimization problems, the accuracy requested of a trust region solver is typically kept fixed. Despite this, however, some informal tests were conducted in which a much more accurate solution was requested of both TRSQPF and GQTPAR. The results indicated that for a given trust region problem of the type considered in this paper, the maximum number of correct significant digits in the trust region step GQTPAR could return was for the most part determined by the value of μ and the machine precision. For example, if μ is 10^{-9} and the machine precision is on the order of 10^{-17}, then in general at most the first 17-9 = 8 digits of the trust region step would be correct. TRSQPF, however, would in general return a solution correct to nearly full precision.

5. Concluding remarks. Our objective in this work has been to develop a technique for the solution of the model trust region problem (1.1) under the assumption that the objective function is the L_2 penalty function. In particular, we have tailored the Moré and Sorensen [1983] algorithm with this specific structure in mind. Our numerical experiments indicate that it can be beneficial to use this modified algorithm. This is particularly true when increased accuracy is demanded, the penalty parameter is quite small, or the current quadratic exhibits a saddle point. It should be noted that in this context algorithm GQTPAR requires additional preprocessing since the product AA^T must be explicitly determined. On the other hand, TRSQPF requires computation of the Bunch-Parlett factorization of the extended system that is more expensive than the Cholesky factorization of the Hessian matrix.

With respect to the method presented in this paper there are several avenues of research we intend to pursue. Among the most obvious possible things we could do is to extend the existing method to the sparse case. At first glance it might seem that the technology used to exploit sparsity in the Cholesky factorization could be adapted to the Bunch-Kaufman factorization. The latter, however, utilizes a complicated pivoting scheme that is required to guarantee stability. It is not clear that we can exploit spartsity and maintain stability in the Bunch-Kaufman factorization. Another possible endeavor would be to develop an implementation of the Bunch-Kaufman factorization on a parallel architecture. Here again the pivoting scheme would no doubt hinder our efforts. Throughout this paper we have assumed that the matrix A is of full column rank. The case where A is rank deficient (or nearly rank deficient) will be the focus of some future work.

The most important remaining project, and the one we plan to dispatch first, is to effectively use the algorithm proposed in this paper in a global method for solving equality constrained optimization problems. Existing methods suffer from often having to decrease μ slowly and consequently can be inefficient. Our goal is to overcome such problems.

Appendix A. We shall now restate and prove Lemma 3.11.

LEMMA 3.11. *If* $\lambda \to -\lambda_1^+$, *then for the* $\hat{z}$ *from Algorithm* 3.10, $\hat{z}^T(H+\lambda I)\hat{z} \to 0$.

Proof. The proof is basically a suitable generalization of the proof given by Moré and Sorensen [1983]. Unfortunately, our use of the extended system and the Bunch-Kaufman factorization complicates matters. Therefore, to facilitate understanding the proof we begin by outlining the proof strategy.

(I) Show that as $\lambda \to -\lambda_1^+$ and $H+\lambda I$ becomes singular, then the extended system X (equation (2.9)) also becomes singular.

(II) For the vector $\hat{y} = y/\|y\|$ (where y is computed by Algorithm 3.10), prove that $\hat{y}^T X \hat{y} \to 0$ as $\lambda \to -\lambda_1^+$.

(III) If $y^T = (z_1^T, z_2^T)$ where $z_1 \in R^n$ and $z_2 \in R^t$ and $\hat{z} = z_1/\|z_1\|$, show that $\hat{z}^T(H+\lambda I)\hat{z} \to 0$ as $\lambda \to -\lambda_1^+$.

Let v_1 be an eigenvector of H corresponding to λ_1. The $(H+\lambda I)v_1 = (\lambda_1+\lambda)v_1$ and so $\|(H+\lambda I)v_1\| \to 0$ as $\lambda \to -\lambda_1^+$. We now observe

$$\text{(A1)} \qquad \begin{pmatrix} B+\lambda I & A \\ A^T & -\mu I \end{pmatrix} \begin{pmatrix} v_1 \\ (1/\mu)A^T v_1 \end{pmatrix} = \begin{pmatrix} (\lambda_1+\lambda)v_1 \\ 0 \end{pmatrix}.$$

This shows that some singular value of X tends to zero as $\lambda \to -\lambda_1$. Since $\|X\|_2$ is bounded in a neighborhood of $-\lambda_1$ we may conclude that the determinant of X goes to zero, and hence X becomes singular, as $\lambda \to -\lambda_1$.

The proof in this section relies heavily on the results of Bunch and Kaufman [1977]. They show that if $\tilde{M} = \tilde{U}S^{-1}$ then $|\tilde{M}_{ij}| \leq 1/(1-.6404)$, and if $\tilde{D} = SDS$ then $|\tilde{D}_{ij}| \leq (2.57)^{n+t-1} \max_{1 \leq i,j \leq n+t} |X_{ij}|$. Again, since the entries of X are bounded in a neighborhood of $-\lambda_1$, it follows that the entries of $\tilde{U}DS$ are also bounded. The orthogonality of Q further implies that the entries of R remain bounded as $\lambda \to -\lambda_1^+$.

Let $\rho > 0$ satisfy

(A2) $$\sum_{j=1}^{n+t} \sum_{i=1}^{j} |r_{ij}| \leqq \rho.$$

It follows from the way we compute w_k^+ and w_k^- in Algorithm 3.10 that

(A3) $$\max(|w_k^+|, |w_k^-|) \geqq \frac{1}{|r_{kk}|}$$

and for our choice of w_k we now have

(A4) $$\frac{1}{|r_{kk}|} \leqq |w_k| + \sum_{i=1}^{k-1} |\theta_i + r_{ik} w_k|.$$

This together with equation (A2) implies

(A5) $$\frac{1}{|r_{kk}|} \leqq (1+\rho)\|w\| \quad \text{or} \quad \|w\| \geqq \frac{1}{(1+\rho)|r_{kk}|},$$

which holds for all k and so

(A6) $$\frac{1}{\|w\|} \leqq \min(|r_{kk}|: k = 1, \ldots, n+t)(1+\rho) \quad \text{or} \quad \|w\| \geqq \frac{1}{\min(|r_{kk}|)(1+\rho)}.$$

Now since the entries of $S^{-1}\tilde{U}^T$ are bounded there exists a $\delta > 0$ such that $\|S^{-1}\tilde{U}^T\|_2 \leqq \delta$. This along with equation (A6) and the orthogonality of P now imply

$$\|y\| = \|x\| \geqq \frac{1}{\delta}\|w\| \geqq \frac{1}{\delta(1+\rho)\min(|r_{kk}|)}.$$

If $\hat{y} = y/\|y\|$ then

$$\hat{y}^T X \hat{y} = \frac{\hat{y}^T e}{\|y\|} \leqq \hat{y}^T e[\delta(1+\rho)\min(|r_{kk}|)]$$

and by the Cauchy-Schwartz inequality

$$\hat{y}^T X \hat{y} \leqq (n+t)^{1/2} \delta(1+\rho)\min(|r_{kk}|).$$

It remains to show that $\min(|r_{kk}|) \to 0$ as $\lambda \to -\lambda_1^+$. To do so we must investigate, in some detail, the pivoting strategy and the choice of entries for S used by Bunch and Kaufman [1977]. For $l = n+t, \cdots, 1$, let $A_{ij}^{(l)}$ denote the ijth entry in the $l \times l$ unfactored matrix remaining at the lth iteration of the Bunch-Kaufman factorization. At this point S_{ll} is chosen according to the following rule:

$$S_{ll} = \begin{cases} 1 & \text{if } (|A_{ll}^{(l)}| \geqq \lambda\lambda^{(l)}) \text{ or } (|A_{rr}^{(l)}| \geqq \lambda\sigma^{(l)}) \text{ or } (\lambda^{(l)} \geqq \sigma^{(k)}), \\ \min(\sqrt{\lambda\sigma^{(1)}/|A_{ll}^{(l)}|}, \sigma^{(l)}/\lambda^{(l)}) & \text{otherwise} \end{cases}$$

where $\lambda = .6404$ and $\lambda^{(l)}$ is the absolute value of the largest off-diagonal element in the last column of $A^{(l)}$. That is,

$$\lambda^{(l)} = \max_{1 \leqq i < l} |A_{il}^{(l)}|$$

and if r is the least integer such that $|A_{rl}^{(l)}| = \lambda^{(l)}$ then $\sigma^{(l)}$ is the largest off-diagonal element in the rth column of $A^{(l)}$,

$$\sigma^{(l)} = \max_{\substack{1 \leqq m \leqq l \\ m \neq r}} |A_{mr}^{(l)}|.$$

It should be noted that $S_{ll} \neq 1$ only when a 2×2 pivot is used or when $\sigma^{(l)} > \lambda^{(l)}$, in which case a 1×1 pivot is used but no permutation is applied to $A^{(l)}$. We now observe that since $\tilde{U}$ is a product Gauss transformations, it has determinant one. It follows that $|\det X| = |\det D|$ and $|\det R| = |\det DS|$ and hence if the determinant of X is small then so is the determinant of D. More specifically, either one of the 1×1 blocks of D must be small, or one of the 2×2 blocks must have a small determinant. For the case where $S_{ll} \neq 1$ and $\sigma^{(l)} > \lambda^{(l)}$ then a 1×1 pivot is used and $D_{ll} = A_{ll}^{(l)}$. Furthermore,

$$(DS)_{ll} \leqq \sqrt{\lambda \sigma^{(l)} |A_{ll}^{(l)}|}$$

and since $\sigma^{(l)}$ and λ are bounded, this shows that if a 1×1 block in D has small magnitude then so must the corresponding entry in DS. The case where $S_{ll} \neq 1$ and a 2×2 block is used is more complicated. When this happens

$$\begin{pmatrix} D_{l-1,l-1} & D_{l-1,l} \\ D_{l,l-1} & D_{l,l} \end{pmatrix} = \begin{pmatrix} A_{rr}^{(l)} & \lambda^{(l)} \\ \lambda^{(l)} & A_{ll}^{(l)} \end{pmatrix}$$

and the conditions of the pivoting scheme imply

$$-\lambda^{(l)^2}\left[\frac{\lambda |A_{rr}^{(l)}|}{\sigma^{(l)}} + 1\right] \leqq A_{ll}^{(l)} A_{rr}^{(l)} - \lambda^{(l)^2} \leqq -(1-\lambda^2)\lambda^{(l)^2} \leqq 0$$

where the center term is the determinant of the 2×2 pivot above. The important implication of equation (A18) is that if the center term is small then so is $\lambda^{(l)}$. Moreover, if the last column of the 2×2 pivot is scaled by S_{ll} then the determinant of the new 2×2 block satisfies

$$S_{ll} |A_{ll}^{(l)} A_{rr}^{(l)} - \lambda^{(l)^2}| \leqq \lambda^{(l)} [\lambda |A_{rr}^{(l)}| + \sigma^{(l)}].$$

Taken together, these observations imply that $|\det DS| \to 0$ as $|\det D| \to 0$ and by virtue of equation (A15) we now know that $\min(r_{kk}) \to 0$ as $\lambda \to -\lambda_1^+$. This completes the proof of the second part.

To prove the last part we recall that $Xy = PQe$ and if we let $PQe = (u_1, u_2)^T$, then

$$\begin{pmatrix} B+\lambda I & A \\ A^T & -\mu I \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \tag{A7}$$

from which it follows that

$$\hat{z}^T (H + \lambda I) \hat{z} = \left(\hat{z}^T u_1 + \frac{1}{\mu} \hat{z}^T A u_2 \right) \Big/ \|z_1\|.$$

It is easy to see that the numerator of the second term is bounded and hence all we need to show is that $\|z_1\|$ must grow as $\|y\|$ becomes large. Multiplying out equation (A7) gives

$$(B+\lambda I) z_1 + A z_2 = u_1$$

and since A has full column rank and $\|u_1\|$ is bounded, we deduce that $\|z_1\| \to \infty$ as $\|z_2\| \to \infty$. This completes the proof. □

REFERENCES

J. R. BUNCH AND L. KAUFMAN [1977], *Some stable methods for calculating inertia solving symmetric linear systems*, Math. Comp., 31, pp. 163-179.

J. E. DENNIS, JR. AND R. B. SCHNABEL [1983], *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ.

J. J. DONGARRA, J. R. BUNCH, C. B. MOLER, AND G. W. STEWART [1979], *Linpack Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA.

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

N. I. M. GOULD [1986], *On the accurate determination of search directions for simple differentiable penalty functions*, I.M.A. J. Numer. Anal., 6, pp. 357-372.

M. D. HEBDEN [1973], *An algorithm for minimization using exact second derivatives*, Atomic Energy Research Establishment report T.P. 515, Harwell, England.

J. J. MORÉ AND D. C. SORENSEN [1983], *Computing a trust region step*, SIAM J. Sci. Statist. Comput., 4, pp. 553-572.

C. H. REINSCH [1967], *Smoothing by spline functions*, Numer. Math., 10, pp. 177-183.

——— [1971], *Smoothing by spline functions* II, Numer. Math., 16, pp. 451-454.

D. C. SORENSEN [1982], *Newton's method with a model trust region modification*, SIAM J. Numer. Anal., 19, pp. 409-426.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 2, pp. 203-229, March 1990

SOLVING THE SYMMETRIC TRIDIAGONAL EIGENVALUE PROBLEM ON THE HYPERCUBE*

ILSE C. F. IPSEN† AND ELIZABETH R. JESSUP†

Abstract. This paper describes implementations of Cuppen's method, bisection, and multisection for the computation of all eigenvalues and eigenvectors of a real symmetric tridiagonal matrix on a distributed-memory hypercube multiprocessor. Numerical results and timings for Intel's iPSC-1 are presented. Cuppen's method is found to be the numerically most accurate of three methods, while bisection with inverse iteration is observed experimentally to be the fastest method.

Key words. hypercube, symmetric tridiagonal eigenvalue problem, Cuppen's method, bisection, multisection, inverse iteration

AMS(MOS) subject classifications. 65F15, 65W05

1. Introduction. Several algorithms have been conceived specifically for determining eigenvalues and eigenvectors of real, symmetric, tridiagonal matrices on conventional uniprocessors. These include the shifted QR algorithm [5], divide-and-conquer strategies [7], [16], and bisection based on Sturm sequence evaluations coupled with inverse iteration [23]. QR and Cuppen's divide-and-conquer method are often used to compute all eigenvalues and eigenvectors of the matrix, while bisection with inverse iteration is normally used when only a few of the eigenvalues and corresponding eigenvectors are required. Of these schemes, the shifted QR algorithm alone does not seem to have an efficient parallel implementation: the shift computation and the application of rotations cannot be overlapped. In contrast, Cuppen's method and the generalization of bisection known as multisection have been implemented with significant speedups on shared-memory multiprocessor architectures and their simulators [4], [8], [17] and on the grid-based, bit-sliced ICL DAP [1], [2]. In this paper, we investigate the suitability of the distributed-memory hypercube architecture as represented by Intel's iPSC-1 for the parallel solution of the symmetric tridiagonal eigenvalue problem.

Three methods for solving the symmetric tridiagonal eigenproblem on a hypercube multiprocessor are presented. The first is a parallel version of Cuppen's method; the second is bisection together with inverse iteration, and the third is multisection with inverse iteration. Unlike most previous parallel eigensolvers, the present implementations operate on a multiprocessor architecture in which each processor has direct access to its own local memory only. Without common memory, exchange of data between processors is accomplished through message passing. An algorithm is implemented in parallel, in general, by dividing the work required into parts or *tasks*, some or all of which can be executed simultaneously. On a shared-memory machine, tasks can be maintained in a queue and the task at the head of the queue is allotted to the first available processor. In a message-passing environment, the assignment of one processor as a queue-manager represents a potential bottleneck. Instead, processors pass messages

* Received by the editors September 14, 1987; accepted for publication (in revised form) March 8, 1989. This work was partially supported by Office of Naval Research contracts N00014-82-K-0184 and N00014-85-K-0461 and by Army Research Office contract DAAL03-86-K-0158. Part of this work was performed while the authors were in residence at the Computer Science and Systems Division of AERE Harwell, United Kingdom, and while the second author was in residence at the Mathematics and Computer Science Division of Argonne National Laboratory, Argonne, Illinois 60810.

† Department of Computer Science, Yale University, P.O. Box 2158, New Haven, Connecticut 06520.

to inform one another about the progress of their tasks and, thereby, coordinate further task allocation.

In the implementations to be discussed, tasks are assigned *statically* to the processors, that is, the processor assigned to each task is known in advance. This strategy permits simplicity of programming and reduction of scheduling overhead. Nevertheless, the cost of communication between processors is often nonnegligible. For this reason, tasks must be apportioned so that communication does not occupy a significant portion of the total computation time. Solution of this scheduling problem (i.e., the partitioning of the algorithm into tasks and assigning of the tasks to processors) is the basis for development of an efficient hypercube program.

This paper's presentation of the three eigensolvers is organized as follows. Section 2 is an introduction to the hypercube multiprocessor employed for numerical experiments and to the notation employed in analysis. Algorithms for data transmission and distributed matrix computation are described in § 3. Cuppen's method, bisection, and multisection are outlined in §§ 4, 5, and 6, respectively; timings for the methods on the Intel iPSC-1 are presented in the respective sections. The paper concludes with a comparison of the three methods in § 7.

2. Some preliminaries. The eigenvalue codes were implemented on an Intel iPSC-1/d5M hypercube multiprocessor with scalar processors. This machine consists of 32 identical node processors each capable of communicating directly with five neighboring processors. A node can communicate with only one of its neighbors at a time and does so by issuing a *send* communication primitive to initiate a message transfer or a *receive* primitive to intercept one. Messages arriving at a node are held in a queue until selected via a receive command. Each processor has 4.5 Mbytes of local memory. For these implementations, the additional, separate host processor is used only for downloading code onto, passing initial data to, and accumulating final results from the node processors.

For purposes of estimating computation times on the hypercube, it is assumed throughout this paper that time $\beta + k\tau$ is required to send a vector of length k from one processor to a neighbor, where β is the communication startup time and τ is the time to transfer one vector element. As in [12], the time required to perform a floating point operation (*flop*) of the form

$$c_{ij} = c_{ij} + a_{ik}b_{kj}$$

is denoted by ω, it includes the time for a floating point multiplication and addition as well as for some pointer manipulation. If the array elements are real double precision floating point numbers, then $\beta/\omega \approx 10$ and $\beta/\tau \approx 125$ on the iPSC-1 running operating system release R3.0.

In our implementations, matrices are often stored by assigning an entire column or an entire row to one processor. For simplicity of presentation, it is assumed that the number of processors p divides the order N of the matrix. In the actual implementation, the rows or columns are assigned to processors in such a way that no processor contains more than $\lceil N/p \rceil$ of them.

3. Three basic algorithms. For all three eigensolvers, communication between processors occurs only as a part of global data exchange, matrix multiplication, or orthogonalization by means of the modified Gram–Schmidt method. These three procedures are described below. Their efficient implementation appeals to topological properties of the hypercube interconnection described in detail in [20], [24].

3.1. Data transmission on the hypercube. In the *Alternate Direction Exchange Algorithm* (ADE) [19] each processor in an i-cube sends a vector of length k to all other processors in that cube. The total time to complete the exchange is

$$T^i_{\mathrm{ADE}} = 2[i\beta + (2^i - 1)k\tau].$$

The 2^{d-i} subcubes of dimension i in a d-cube can simultaneously perform such an exchange without interference.

3.2. Matrix multiplication on the hypercube. Our implementations take advantage of the fact that physically interconnected processors in a hypercube form a ring and that subcubes correspond to contiguous sequences within this ring. In a d-cube, the processor identifiers are generated according to a binary reflected Gray code sequence [11], [18], and processor j constitutes the jth member in this sequence, $0 \leqq j \leqq 2^d - 1$. Because a Gray code sequence is cyclic, it defines a *ring* of physically interconnected processors within the hypercube. Within the ring, processor j communicates with its neighboring processors $j-1$ and $j+1$, where the processor identifiers in a d-cube are taken modulo $p = 2^d$.

When using an embedded ring, a matrix may be stored in the hypercube by situating blocks of adjacent columns in neighboring processors. The algorithm *Ring Matrix Multiplication* (RMM) performs the multiplication $C = AB$ of two $N \times N$ matrices A and B both distributed by block columns among the processors of a d-cube, where $N = k2^d$.

Initially, processor j, $0 \leqq j \leqq 2^d - 1$, contains columns $jk, \cdots, (j+1)k-1$ of A and of B and, upon completion, columns $jk, \cdots, (j+1)k-1$ of A, B, and C. During the formation of C, the columns of B remain in their original places while the columns of A are passed along the ring from processor to processor, overwriting the previously held columns of A in each processor. Let $\hat{B}_{ij}$ denote the $k \times k$ block-matrix in position (i, j) of B, $0 \leqq i, j \leqq 2^d - 1$, and the vectors

$$\hat{B}_j = [\hat{B}^T_{0j} \cdots \hat{B}^T_{2^d-1,j}]^T$$

be the k columns $jk, \cdots, (j+1)k-1$ of $\hat{B}$. In Algorithm RMM below, indices should be taken modulo 2^d.

```
ALGORITHM RMM.
In parallel, do on all processors $j$, $0 \leqq j \leqq 2^d - 1$:
    $\hat{C}_j = 0$
    For $i = 0, \cdots, 2^d - 1$:
    (1) compute $\hat{C}_j = \hat{C}_j + \hat{A}_{j-i}\hat{B}_{j-i,j}$
    (2) send $\hat{A}_{j-i}$ to processor $j+1$
    (3) receive $\hat{A}_{j-i-1}$ from processor $j-1$
```

The arithmetic time at each iteration is $2^d k^3 \omega$, and the communication time is $\beta + 2^d k^2 \tau$, giving a total time of

$$T^d_{\mathrm{RMM}} = 2^{2d} k^3 \omega + 2^d(\beta + 2^d k^2 \tau).$$

See also [10].

3.3. Modified Gram–Schmidt orthogonalization. The *Modified Gram-Schmidt* (MGS) procedure transforms a set of linearly independent vectors into a set of orthonormal vectors. This is necessary, for example, when inverse iteration applied to poorly separated eigenvalues produces eigenvectors that, while linearly independent, are not orthogonal [23].

Algorithm MGS overwrites a set of m linearly independent vectors $\{v_0, \cdots, v_{m-1}\}$ of length N with an orthonormal set spanning the same space. First consider the case $m \le p$, where v_j resides in processor j. The kth orthonormalized vector is computed during the kth step of MGS. The orthonormalized vectors are passed from processor to processor to effect the orthonormalization of the remaining vectors.

```
ALGORITHM MGS ($m \leqq p$).
In parallel, do on all processors $j$, $0 \leqq j \leqq m-1$:
    (1) for $k = 0, \cdots, j-1$
        (1.1) if $j > k$, receive $k$th vector $v_k = (v_{1k}, \cdots, v_{Nk})^T$ from processor $j-1$
        (1.2) if $j < m-1$, send $v_k$ to processor $j+1$
        (1.3) compute $r_{kj} = \sum_{i=1}^N v_{ik} v_{ij}$
        (1.4) for $i = 1, \cdots, N$, $v_{ij} = v_{ij} - v_{ik} r_{kj}$
    (2) normalize $j$th vector: $(v_{1j}, \cdots, v_{Nj})^T = (v_{1j}, \cdots, v_{Nj})^T / \|(v_{1j}, \cdots, v_{Nj})^T\|_2$
    (3) if $j < m-1$, send $v_j$ to processor $j+1$
```

Processor j requires access to the orthonormalized vectors $v_0, \cdots, v_{j-1}$ in order to orthogonalize v_j. Processor j must therefore remain idle until v_0 has been normalized and forwarded from processor 0. Passing v_0 along the j communication links requires time $j(\beta + N\tau)$. Once it has used v_0 for the orthonormalization of v_j, processor j must wait an additional $2N\omega$ steps for v_1 to arrive. This additional idle time is equal to the time needed to normalize v_1. Thus, processor j waits a total time of $2N\omega$ for each of the vectors $v_0, \cdots, v_{j-1}$. When $\beta + N\tau < 2N\omega$, the data transfer from processors $1, \cdots, j-1$ to processor j overlaps this additional idle time and need not be considered further. For the iPSC-1, $\beta + N\tau < 2N\omega$ for $N > 5$. The total time for Algorithm MGS is equal to the time needed by the processor holding the last column, so

$$T_{\text{MGS}} = 2(2m-1)N\omega + (m-1)(\beta + N\tau) \quad \text{for } m \leqq p.$$

In the case $m > p$, processor j contains $v_j, v_{j+p}, \cdots, v_{j+\nu p}$ where $j + \nu p \leqq m-1$, and processors are connected in a ring. Communication takes place in a manner similar to that of Algorithm MGS so that no processor encounters any column more than once. Hence, no processor waits for v_0 for more than $(p-1)(\beta + N\tau)$ steps. Upon receipt of v_k, processor j performs steps (1.3) and (1.4) for vectors $v_j, \cdots, v_{j+\nu p}$ with index greater than k. As soon as step (1) has been completed for v_j, that vector is normalized and sent out so that one normalized vector begins the circuit every $4N\omega$.

The time to complete MGS is again equal to the total time required by the processor holding v_{m-1}. A lower bound for the case $m = kp$, $k \geqq 1$, and $\beta + N\tau < 2N\omega$ is given by the time required for processor $p-1$ to receive v_0 and to orthogonalize all its vectors

$$T_{\text{MGS}} \geqq \left((m-1)\frac{m}{p} + m + 1\right) 2N\omega + (p-1)(\beta + N\tau) \quad \text{for } m = kp.$$

4. Cuppen's method.

The first scheme to be discussed is a divide-and-conquer method described by Cuppen to find all the eigenvalues and corresponding eigenvectors of any symmetric tridiagonal matrix [7].

4.1. Sequential algorithm.

The method is based on the fact that a symmetric tridiagonal matrix T of even order N can be divided into a pair of equal-sized symmetric tridiagonal submatrices as follows:

$$T=\begin{pmatrix} T_0 & \\ & T_1 \end{pmatrix}+\theta\alpha\begin{pmatrix} e_{N/2} \\ \theta^{-1}e_1 \end{pmatrix}(e_{N/2}^T \quad \theta^{-1}e_1^T) \tag{4.1}$$

where α is the off-diagonal element of T at position $N/2$; e_i is the ith unit vector of length $N/2$, and T_0 and T_1 are of order $N/2$. The sign and magnitude of θ are selected to ensure that subdivision of the matrix can be performed at this position without cancellation [8]. The original problem has now been split into two eigenproblems of half its order.

If the solutions to the two smaller eigensystems are $T_0=X_0D_0X_0^T$ and $T_1=X_1D_1X_1^T$, then

$$T=Q\left[D+\alpha\theta\begin{pmatrix} l_0 \\ \theta^{-1}f_1 \end{pmatrix}(l_0^T \quad \theta^{-1}f_1^T)\right]Q^T$$

where

$$Q=\begin{pmatrix} X_0 & \\ & X_1 \end{pmatrix}, \qquad D=\begin{pmatrix} D_0 & \\ & D_1 \end{pmatrix}.$$

$l_0^T=e_{N/2}^TX_0$ is the last row of X_0, and $f_1^T=e_1^TX_1$ is the first row of X_1. To solve the eigenproblem for T, it is necessary to find the eigenvalues and eigenvectors of the diagonal matrix D plus a rank-one correction, $D+\rho zz^T=Q^TTQ$, where $z^T=\sqrt{\alpha\theta/\rho}(l_0^T \quad \theta^{-1}f_1^T)$, and ρ is selected so that $\|z\|_2=1$. For the remainder of the paper, it is assumed that T is unreduced (i.e., its off-diagonal elements are nonzero). If not, T would consist of a direct product of disjoint, lower-order matrices whose eigensolutions could be determined independently.

The eigenvalues $\lambda_0,\cdots,\lambda_{N-1}$ of $D+\rho zz^T$ are found using a root-finding technique developed in [6]. The corresponding eigenvectors are computed from

$$u_i=\frac{(D-\lambda_iI)^{-1}z}{\|(D-\lambda_iI)^{-1}z\|_2}. \tag{4.2}$$

If U is the matrix with columns $u_0, u_1, \cdots, u_{N-1}$ and $\Lambda=\operatorname{diag}(\lambda_0,\cdots,\lambda_{N-1})$, the eigendecomposition of the original matrix may now be expressed as the matrix product

$$T=QU\Lambda U^TQ^T.$$

If the elements of D are not sorted in descending order as is required for the root-finding procedure mentioned above, it is necessary to consider instead the matrix J^TDJ, where J is an appropriate permutation matrix. Distinctness of the elements $\{d_j\}$ is also requisite. Although an unreduced tridiagonal matrix of order N does itself have N distinct eigenvalues, the intermediate matrix D may not. For example, if

$$T=\begin{pmatrix} 2 & 1 & 0 & 0 \\ 1 & 3 & 1 & 0 \\ 0 & 1 & 3 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix},$$

then $\alpha=1$,

$$T_0=\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \quad\text{and}\quad T_1=\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}.$$

Both T_0 and T_1 have eigenvalues 1 and 3, so

$$D=\begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

and $d_0=d_1$, $d_2=d_3$.

In the case of exact arithmetic, the deflation rules may be described as follows. When multiple values $d_l=d_{l+1}=\cdots=d_{l+k}$ occur, the eigenvector basis is first rotated to zero out the elements $z_{l+1},\cdots,z_{l+k}$ corresponding to the multiple elements $d_{l+1}=\cdots=d_{l+k}$: a product of plane rotations G_l is applied so that

$$G_l(z_l, z_{l+1},\cdots,z_{l+k})^T=(z'_l,0,\cdots,0)^T.$$

The eigenvalue corresponding to a zero element z'_j of the rotated vector remains unchanged ($\lambda_j=d_j$), and the corresponding eigenvector may be chosen as the appropriate unit vector of order N ($u_j=e_j$) [6]. Therefore, multiple values along the diagonal of D result in a significant reduction in the work required to compute the eigensystem of $D+\rho zz^T$. Zero elements of z corresponding to distinct elements of D lead to similar savings. Deflation rules have been developed for finite precision in [8]: rotations are applied when close elements of D occur, and deflation occurs when elements of z are small. The same rules are employed for the hypercube implementation. Numerical experiments have confirmed that the increase in speed due to this deflation is substantial [8]. Representing the product of all rotations by the matrix G, the matrix T is expressed as

$$T=QJG^TU\Lambda U^TGJ^TQ^T=X\Lambda X^T \tag{4.3}$$

where $U\Lambda U^T$ is the eigendecomposition of $GJ^T(D+\rho zz^T)JG^T$. The eigenvalues of T are the diagonal elements of Λ while the eigenvectors of T are the columns of $X=QJG^TU$.

4.2. Parallel algorithm. The parallel implementation of Cuppen's method recursively continues the divide-and-conquer strategy. The matrix T is written as

$$T\equiv T_{20}=\left[\begin{array}{c|c} \left(\begin{array}{c|c} T_{00} & 0 \\ \hline 0 & T_{01}\end{array}\right)+\alpha_{10}b_{10}b_{10}^T & 0 \\ \hline 0 & \left(\begin{array}{c|c} T_{02} & 0 \\ \hline 0 & T_{03}\end{array}\right)+\alpha_{11}b_{11}b_{11}^T \end{array}\right]+\alpha_{20}b_{20}b_{20}^T. \tag{4.4}$$

This subdivision or "tearing" process is repeated and rank-one updating procedures applied recursively. At the ith subdivision, a submatrix T_{ij} is split into

$$T_{ij}=\begin{pmatrix} T_{i-1,2j} & \\ & T_{i-1,2j+1}\end{pmatrix}+\alpha_{ij}b_{ij}b_{ij}^T,\qquad 0\leqq j\leqq 2^{d-i}-1.$$

It is this recursive nature of Cuppen's method that suggests its suitability to parallel implementation on the hypercube architecture. A high-level description of a parallel implementation of Cuppen's method on the hypercube is given as Algorithm C1.

ALGORITHM C1. Solution of eigenproblem of order $N=k2^d$ on a d-cube.

Recursively divide the matrix $T\equiv T_{d0}$ d times so that processor j contains submatrix T_{0j} of order k, $0\leqq j\leqq 2^d-1$.

(1) *Step* 0. Each processor (0-cube) independently computes the eigensystem of its order-k submatrix.

(2) *Step* i, $1 \leqq i \leqq d$. Each $(i-1)$-cube pairs with another $(i-1)$-cube to form an i-cube by exchanging information about the eigensystems of the matrices $T_{i-1,2j}$ and $T_{i-1,2j+1}$ of order $k2^{i-1}$ computed in Step $i-1$. Each i-cube independently computes the eigensystem for its matrix T_{ij} of order $k2^i$. The 2^{d-i} i-cubes at Step i are enumerated by the index j, $0 \leqq j \leqq 2^{d-i}-1$.

The number of updating steps in Algorithm C1 is equal to the dimension of the hypercube. At Step i, 2^{d-i} i-cubes independently solve eigensystems of order $k2^i$. Upon completion of Step d, each processor contains k of the $N = k2^d$ eigenvalues of the original matrix T as well as the k corresponding eigenvectors (of length N). Algorithm C1 can now be refined to explain in greater detail the assignment of computational tasks to processors.

Each processor starts with the solution of an eigensystem of order k and during subsequent steps is responsible for updating k eigenvalues and k eigenvectors, thereby doubling the length of the vectors during each step. Thus, the jth processor in a subcube contains eigenvalues $\lambda_{jk}, \lambda_{jk+1}, \cdots, \lambda_{(j+1)k-1}$ of the matrix T as well as the corresponding eigenvectors. Figure 1 depicts the communication pattern at each step of the algorithm on a hypercube of dimension 3. The labeled squares denote individual processors (the 0-cubes of Step 0). A box containing 2^i processors indicates that, at Step i, subcubes of dimension i independently compute the needed eigensystems. All processors belonging to the same i-cube must communicate with one another during the solution Step i.

The allocation of submatrices to processors for a 3-cube is given in Table 1. The matrix $T \equiv T_{30}$ of order $N = k2^3$ is recursively divided into eight tridiagonal matrices $T_{00}, T_{01}, \cdots, T_{07}$ of order k, and matrix T_{0j} is assigned to processor j, $0 \leqq j \leqq 7$. In general, the entries T_{ij} are those matrices whose eigensystems are computed in Step i by subcube j. The brackets distinguish the subcubes occupied by each eigensystem.

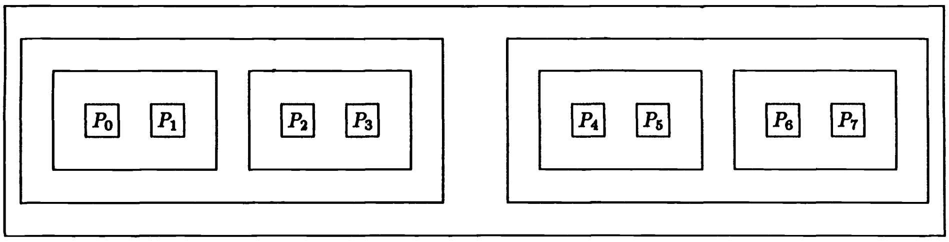


FIG. 1. *Formation of subcubes for data transmission during Cuppen's method.*

TABLE 1
Assignment of submatrices to the processors of a 3-cube during Cuppen's method.

	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
Step 0:	$[T_{00}]$	$[T_{01}]$	$[T_{02}]$	$[T_{03}]$	$[T_{04}]$	$[T_{05}]$	$[T_{06}]$	$[T_{07}]$
Step 1:	$[T_{10}]$		$[T_{11}]$		$[T_{12}]$		$[T_{13}]$	
Step 2:	$[T_{20}]$				$[T_{21}]$			
Step 3:	$[T_{30} \equiv T]$							

More precisely, during Step 0 processor j (a 0-cube) computes the eigensystem (Λ_{0j}, X_{0j}) of the matrix T_{0j}. Because each rank-one updating step requires the eigensystems of two smaller matrices, processors must pair up in Step 1 and exchange information within 1-cubes to compute the eigensystems of the four order-$2k$ matrices $T_{10}, \cdots, T_{13}$. After exchanging information about the eigensystems of matrices T_{00} and T_{01}, processors 0 and 1 can together compute the eigensystem (Λ_{10}, X_{10}) of T_{10} (see Table 1). Processor 0 is left with the leading k eigenvalues and eigenvectors of T_{10}, while processor 1 holds the trailing k. The remaining steps proceed in a similar fashion until, at the end of Step 3, each of the eight processors contains k eigenvalues and k eigenvectors of length $8k$ of the original matrix $T \equiv T_{30}$. For $0 \leqq j \leqq 7$, processor j holds eigenvectors indexed $jk, \cdots, (j+1)k-1$.

To begin the solution of the eigenvalue problem by Cuppen's method, each node requires a sequence of diagonal and off-diagonal elements of the matrix T. Finding the eigensystem of T_{20} in equation (4.4) on a 2-cube, for instance, requires the submatrix T_{00} as well as the off-diagonal elements α_{10} and α_{20} to be available in processor 0. Under the assumption that each node contains the needed matrix elements, Algorithm C2 details the steps in the determination of all eigenvalues and eigenvectors of a matrix T of order $N = k2^d$ on a d-cube. Italics distinguish comments from instructions. Note that only parts (i.1) and (i.4) of Step i, $1 \leqq i \leqq d$, require data communication.

ALGORITHM C2. Solution of eigenproblem of order $N = k2^d$ on a d-cube. Recursively divide the matrix $T \equiv T_{d0}$ d times and allocate submatrix T_{0j} and the d appropriate off-diagonal elements to processor j, $0 \leqq j \leqq 2^d - 1$.

Step 0. Processor j (0-cube) computes the eigensystem (Λ_{0j}, X_{0j}) of the matrix T_{0j} of order k, the diagonal of Λ_{0j} contains the eigenvalues of T_{0j} in descending order, and the columns of X_{0j} are the corresponding eigenvectors, $0 \leqq j \leqq 2^d - 1$.

Step $1 \leqq i \leqq d$. *{2^{d-i} i-cubes independently compute the eigensystems (Λ_{ij}, X_{ij}) of the matrices T_{ij} of order $k2^i$ using the eigensystems from Step $i-1$:*

$$T_{ij} = \begin{pmatrix} T_{i-1,2j} & \\ & T_{i-1,2j+1} \end{pmatrix} + \alpha_{ij} b_{ij} b_{ij}^T$$

$$= \begin{pmatrix} X_{i-1,2j} & \\ & X_{i-1,2j+1} \end{pmatrix} \left[\begin{pmatrix} \Lambda_{i-1,2j} & \\ & \Lambda_{i-1,2j+1} \end{pmatrix} + \rho_{ij} z_{ij} z_{ij}^T \right] \begin{pmatrix} X_{i-1,2j} & \\ & X_{i-1,2j+1} \end{pmatrix}^T .$$

Denote by S a subcube of dimension i and index j which consists of processors $j2^i$ through $(j+1)2^i - 1$. For simplicity, denote these processors by $P_0, P_1, \cdots, P_{2^i-1}$ and replace subscripts of the form $(i-1, 2j)$ with 0 and $(i-1, 2j+1)$ with 1. With the new notation, processors $P_0, P_1, \cdots, P_{2^i-1}$ compute the eigensystem (Λ, X) of

$$T = \begin{pmatrix} T_0 & \\ & T_1 \end{pmatrix} + \alpha b b^T = \begin{pmatrix} X_0 & \\ & X_1 \end{pmatrix} \left[\begin{pmatrix} \Lambda_0 & \\ & \Lambda_1 \end{pmatrix} + \rho z z^T \right] \begin{pmatrix} X_0 & \\ & X_1 \end{pmatrix}^T$$

from (Λ_0, X_0) and (Λ_1, X_1). From Step $i-1$, processor P_l for $0 \leqq l \leqq 2^{i-1} - 1$ contains eigenvalues of T_0 indexed $lk, \cdots, (l+1)k-1$ and columns $lk, \cdots, (l+1)k-1$ of X_0. These processors form a subcube of dimension $i-1$ called S_0. Similarly, processor P_l for $2^{i-1} \leqq l \leqq 2^i - 1$ contains eigenvalues of T_1 indexed $lk, \cdots, (l+1)k-1$ and columns $lk, \cdots, (l+1)k-1$ of X_1. These processors form a subcube of dimension $i-1$ called S_1.}

(i.1) By means of Algorithm ADE, processors in S_0 and S_1 exchange their k elements of Λ_0 and Λ_1, respectively, and their k elements of the last row of

X_0 or the first row of X_1, respectively, so that each processor in S contains Λ_0, Λ_1, the last row of X_0 and the first row of X_1.

(i.2) Each processor in S:

(i.2.a) computes z and ρ from the last row of X_0, the first row of X_1, and α.

(i.2.b) determines a permutation matrix J by merging the sorted sequences diag (Λ_0) and diag (Λ_1) so that the diagonal elements of

$$\hat{D}=J^TDJ=J^T\begin{pmatrix}\Lambda_0 & \\ & \Lambda_1\end{pmatrix}J$$

are sorted in descending order.

(i.2.c) permutes the elements of z accordingly: $\hat{z}=J^Tz$.

(i.2.d) applies the product of plane rotations G to zero out the elements in z that correspond to close elements in $\hat{D}$.

(i.2.e) identifies small elements of z and deflates.

(i.3) Each processor P_j in S

(i.3.a) computes elements $jk, \cdots, (j+1)k-1$ of Λ (eigenvalues of $\hat{D}+\rho\hat{z}\hat{z}^T$) and the corresponding eigenvectors $u_{jk}, \cdots, u_{(j+1)k-1}$ by root finding and formula (4.2). The eigenpair is (λ_j, e_j), where e_j is the jth unit vector, when $\hat{z}_j$ is small.

(i.3.b) updates the eigenvectors: $(v_{jk}, \cdots, v_{(j+1)k-1})=G^T(u_{jk}, \cdots, u_{(j+1)k-1})$.

(i.4) By means of Algorithm RMM, processors in S_0 and S_1 send their columns of X_0 and X_1, respectively, to all other processors in S so that processor j can determine its k columns of X via

$$(x_{jk}, \cdots, x_{(j+1)k-1})=\left[\begin{pmatrix}X_0 & \\ & X_1\end{pmatrix}J\right](v_{jk}, \cdots, v_{(j+1)k-1}), \qquad 0\leqq j\leqq 2^{d-i}-1.$$

4.3. Analytical and experimental results. As shown in [8], deflation plays an important role in the serial behavior of Cuppen's method. However, its contribution to the time complexity of the algorithm is difficult to assess analytically. In [15], we derive a first-order approximation to an upper bound on the time needed to determine all the eigenvalues and eigenvectors of a symmetric, tridiagonal matrix on the hypercube in the case of no deflation. For a matrix of order $N=k2^d=kp$, the resulting upper bound on the total arithmetic time T_A is given by

$$T_A\leqq\omega\left(\kappa_0\left(\frac{N}{p}\right)^3+\frac{2}{3}\frac{N^3}{p}+2\kappa\frac{N^2}{p}+4N\right).$$

The first cubic term $\kappa_0(N/p)^3$ is the time required to solve the subproblems in Step 0 of Algorithm C2. The second cubic term $\frac{2}{3}N^3/p$ comes from matrix multiplication during eigenvector updates in Step (i.4). The quadratic term $2\kappa N^2/p$ arises from the computation of intermediate eigenvalues and eigenvectors during Step (i.3.a). Communication in each step consists of a pair of alternate direction exchanges plus the modified Algorithm RMM in Step (i.4). Thus the total communication time T_C is bounded above by

$$T_C\leqq\beta(2d^2+d+2p)+\left(\frac{2}{3}N^2+8N-\frac{N^2}{p}-4d\frac{N}{p}\right)\tau.$$

In [15], it is also shown that assigning columns of the eigenvector matrix to processors as in Algorithm C2 results in a lower time complexity than does an assignment of rows to processors. Although deflation occurs for most matrices, its effects are greatly reduced in the hypercube implementation. This result is evident in speedups measured for two matrices having different amounts of deflation. The speedup of Cuppen's method is defined as the time required to solve a problem using the fastest sequential method on one node divided by the time required to solve the same problem by Cuppen's method using $p \geqq 2$ nodes. In this paper, we compute speedups with respect to the fastest sequential method having accuracy comparable to the parallel method in question. (The accuracies of Cuppen's method, bisection, multisection, and inverse iteration are compared in § 7, and Cuppen's method is seen to be more accurate than the other combinations.) Dongarra and Sorensen's implementation of Cuppen's method SESUPD is the fastest available sequential method for finding to *high accuracy* all eigenvalues and orthogonal eigenvectors of a real symmetric, tridiagonal matrix of order larger than 50 [8] and so is employed for speedup computation. For instance, on one processor of the iPSC-1, SESUPD requires 8809.9 seconds, and EISPACK's TQL2 takes 28,341.9 seconds for matrix $[1, 2, 1]$ of order 512.

The first symmetric, tridiagonal test matrix, $[1, \mu, 1]$, has the value 1.0 in each off-diagonal position and the value $\mu \times 10^{-6}$ in the μth diagonal position. The intermediate eigenvalues of $[1, \mu, 1]$ (i.e., the diagonal elements of D) are distinct at each step, and the small ratios of the diagonal elements μ to the off-diagonal elements ensure nonnegligible elements of z. Thus, little or no deflation occurs at each step in the solution process. The second matrix, $[1, 2, 1]$, has all off-diagonal elements equal to 1 and all diagonal elements equal to 2. Henceforth, this matrix is denoted $[1, 2, 1]$. The structure of $[1, 2, 1]$ leads to significant deflation in the root-finding Step (i.3) of Algorithm C2. SESUPD [8] computes the eigensystem of $[1, 2, 1]$ in approximately half the time it needs for $[1, \mu, 1]$ with exact times dependent on matrix order.

Figure 2 shows speedup for 32 processors as a function of matrix order for $[1, 2, 1]$ and $[1, \mu, 1]$. Although near maximal speedup occurs in the case of little deflation, speedup of only about 50 percent is seen when deflation is prevalent. This difference

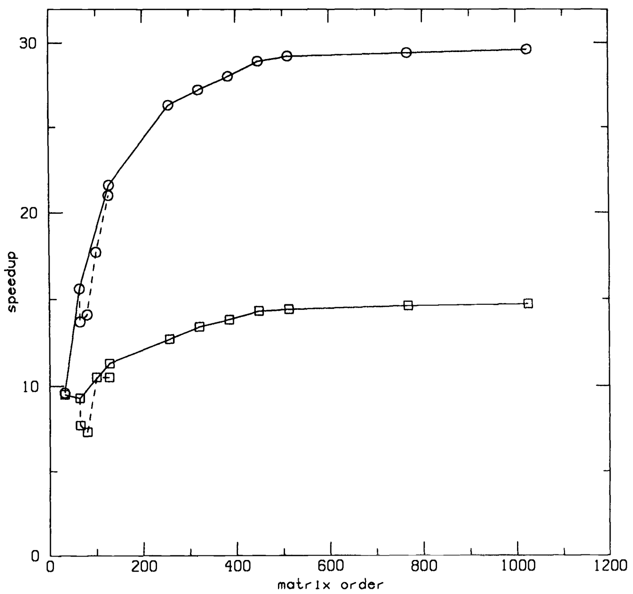


FIG. 2. *Cuppen's method on a 5-cube: speedup for* $[1, 2, 1]$ *(squares) and* $[1, \mu, 1]$ *(circles) versus matrix order. Points for matrix orders that are multiples of* 32 *are connected with solid lines. Other points are connected with dotted lines.*

does not indicate that the eigensystem of $[1, \mu, 1]$ is found in less time than that of $[1, 2, 1]$ on the hypercube, but rather that Cuppen's method on the hypercube does not exhibit the large time savings from deflation shown by SESUPD. On more than one processor, SESUPD requires approximately the same time for $[1, 2, 1]$ as for $[1, \mu, 1]$, but on one processor, SESUPD runs considerably faster for $[1, 2, 1]$ than for $[1, \mu, 1]$. Speedup for $[1, \mu, 1]$ is thus greater.

The loss of the deflation advantage is due in part to the static scheduling of processors and in part to additional overhead incurred in the hypercube implementation. Static scheduling affects the performance because the processors no longer solve identical problems at each step when the cube dimension is larger than one. Nevertheless, the data exchange requirements of Algorithm C2 synchronize the processors. Thus, although a single processor may encounter significant savings when deflation occurs, the gain may not be shared by the cube as a whole. Unless the effects of deflation are evenly distributed over the processors of the cube, any time gained during root-finding by a single processor will be lost as it waits for the slower processors during the data exchange routines. The improvement due to deflation measured for sequential or for shared memory machines, on which root-finding tasks are scheduled dynamically by a queue manager, is not expected for the statically scheduled hypercube. A dynamic scheduler would incur additional overhead and would be difficult to implement without potentially expensive communication of eigenvectors.

The remaining loss of performance is due to communication requirements. At some points in the algorithm, it is more efficient to allow all processors in the cube to perform the same computation than it is to communicate the data required for parallel computation. (Step (i.2.d) of Algorithm C2 is an example.) These redundant computations cause root finding to occupy a smaller fraction of the total computation time on the hypercube than on a sequential machine. Introduction of the communication required for the hypercube similarly reduces the time savings of deflation. Figure 3

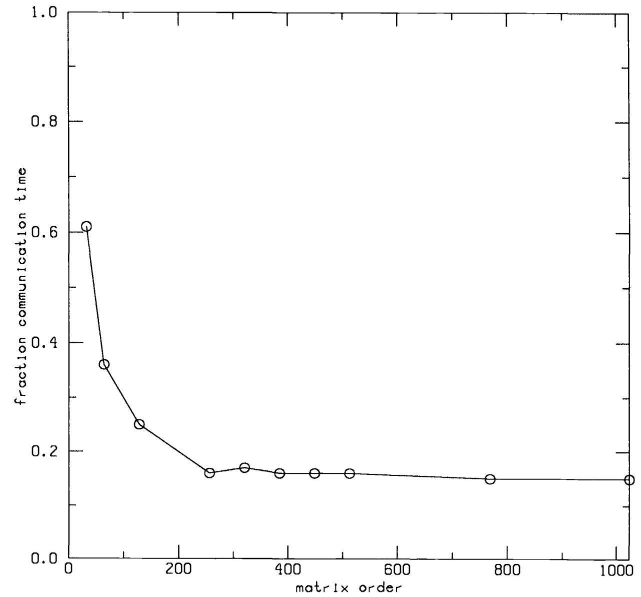


FIG. 3. *Cuppen's method on a 5-cube: fraction of total time spent in communication versus matrix order for matrix* $[1, 2, 1]$.

shows the fraction of time spent in communication by the processors of a 5-cube while determining the eigenvalues and eigenvectors of matrices of various orders for matrix [1, 2, 1]. Computation time is shown to be greater than communication time for matrix orders as small as 64. For eight or more eigenvectors per processor, the communication cost levels off at about 16 percent of the total time.

5. Bisection and inverse iteration.

The method described in this section determines all eigenvalues of a symmetric, tridiagonal matrix from Sturm sequence evaluations at interval endpoints obtained through bisection, and subsequently computes the eigenvectors via inverse iteration.

The bisection procedure recursively halves an initial interval containing all eigenvalues (such as the union of Gerschgorin disks) into two smaller intervals. Division of the interval continues until one of the following three conditions is satisfied: the interval contains no eigenvalue, the interval contains exactly one eigenvalue, or the interval size is smaller than a given tolerance. In the latter case, the interval contains a group of computationally coincident eigenvalues.

In this study, all eigenvalues are determined to full accuracy by bisection using a nonlinear recurrence for the Sturm sequence. Use of a nonlinear recurrence would permit computation of the eigenvalues by such faster methods as the ZEROIN root finder [9] and interpolation [3] but is prone to underflow and overflow. While the use of one of these techniques could improve the performance of the bisection algorithm, the gain in speed would not alter the comparisons presented in the concluding section of this paper. The EISPACK routine TRIDIB [22] was employed for the experiments described below.

The convergence rate of inverse iteration to an eigenvector is inversely proportional to the separation of the associated eigenvalue from the nearest neighboring eigenvalue. Hence, for well-separated eigenvalues, inverse iteration produces orthogonal eigenvectors [23]. In case of close eigenvalues, eigenvalues are first perturbed to a sufficient distance [22], [23] and then the eigenvectors are computed and orthogonalized.

The term *cluster* is used henceforth to denote a group of computationally coincident eigenvalues. If the computed eigenvalues λ_i and λ_{i+1} are computationally coincident, λ_{i+1} is replaced by the value $\lambda_{i+1}+\varepsilon$ where ε is the small multiple of machine epsilon defined in EISPACK's TINVIT [22]. This perturbation of eigenvalues is intended to permit computation of linearly independent eigenvectors by inverse iteration. Because the eigenvalues computed by bisection (or multisection) are highly accurate, the accuracy of the solution is determined by the accuracy of the computed eigenvectors.

5.1. Parallel algorithm.

In our implementation of bisection and inverse iteration, each processor in a d-dimensional hypercube computes k eigenvalues and k eigenvectors of a matrix of order $N=k2^d=kp$. During bisection, processor i computes eigenvalues indexed $ik, \cdots, (i+1)k-1$ by means of the EISPACK routine TRIDIB [22]. Because orthogonalization may be necessary, the computation of eigenvectors requires attention to load balancing. The following three approaches to the distribution of eigenvectors to processors are examined in [15]:

(1) Assigning a block of adjacent eigenvectors to each processor. In this case, effective pipelining in the orthogonalization Algorithm MGS is not possible meaning that processor load imbalance can be severe.

(2) Using a scheduling heuristic to distribute tasks to processors according to a specified cost function. Here, significant overhead is involved, and the resulting assignment of work to processors is not necessarily well balanced.

(3) Assigning eigenvectors to processors in a cyclic (or wrapped) fashion. For this arrangement, processor utilization is well balanced, and the implementation is simple.

We implemented the third, cyclic distribution strategy, where processor j computes eigenvectors indexed $j, j+p, j+2p, \cdots, j+\nu p \leqq N$. As opposed to the second, heuristic scheduling approach, the assignment of eigenvectors to processors is predetermined, and no processor computes more than one eigenvector more than any other processor. Unless a cluster of eigenvalues comprises more than p eigenvalues, no processor handles more than one vector associated with any cluster. Except in the event of a particular distribution of small clusters, the burden of orthogonalization is spread across processors.

The biggest drawback to cyclic distribution is the extent of communication required. Although each cluster requires communication for the orthogonalization of its eigenvectors, the number of messages sent by a processor is essentially independent of the cluster size because the eigenvectors are distributed over as many processors as possible. Moreover, communication and computation are overlapped by Algorithm MGS and by the simultaneous orthogonalization of vectors associated with different clusters.

Because eigenvalues are computed according to a block distribution and eigenvectors according to a cyclic distribution, an intermediate redistribution of eigenvalues (via Algorithm ADE) is necessary to make all eigenvalues available to each processor. Alternatively, the computation of eigenvalues could be assigned to processors in a cyclic fashion so as to obviate the intermediate exchange stage. If that approach were employed, a processor would have to compute eigenvalues in nonadjacent intervals so that either the number of Sturm sequence evaluations would increase or else more messages would have to be sent.

ALGORITHM B: Solution of eigenproblem of order $N=kp$ on a p-processor hypercube. Each processor contains all diagonal and off-diagonal elements of the matrix.

In parallel, all processors perform the following steps.

Step 0. *Determination of initial interval.* Processor i computes all Gerschgorin disks to find the interval containing all N eigenvalues and then uses bisection to determine the interval containing the N/p eigenvalues indexed $ik, \cdots, (i+1)k-1$.

Step 1. *Computation of eigenvalues.* Processor i determines eigenvalues indexed $ik, \cdots, (i+1)k-1$ via bisection.

Step 2. *Exchange of eigenvalues.* All processors exchange their eigenvalues by means of Algorithm ADE.

Step 3. *Perturbation of eigenvalues.* Each processor sorts all N eigenvalues and perturbs eigenvalues belonging to clusters.

Step 4. *Computation of eigenvectors.* Processor i computes the N/p eigenvectors corresponding to eigenvalues indexed $i, i+p, \cdots, i+\nu p \leqq N$ by inverse iteration.

Step 5. *Orthogonalization of eigenvectors.* Each processor orthogonalizes those of its eigenvectors that are associated with a cluster by Algorithm MGS.

5.2. Experimental results. The efficiency of our implementation is reflected in Fig. 4 where speedup is plotted against matrix order for matrix [1, 2, 1] and for random matrices. (Because different random matrices were generated for each order, there is no relation between them.) The speedup is calculated as the time for a single processor

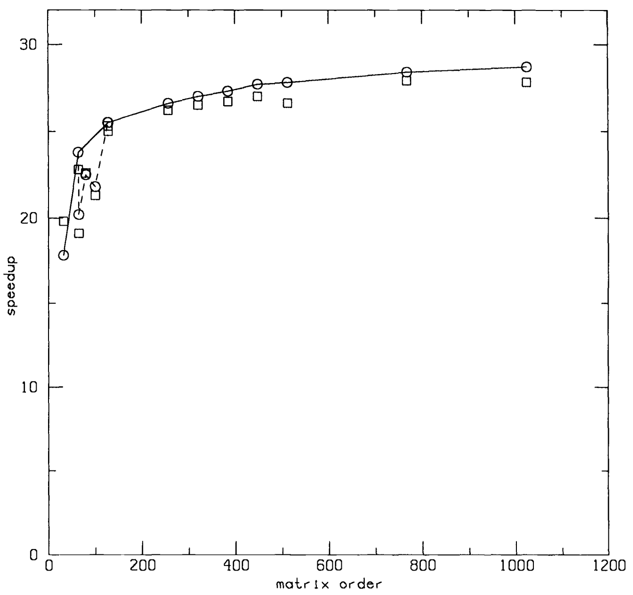


FIG. 4. *Bisection on a 5-cube: speedup for* [1, 2, 1] *(circles) and random matrices (squares) versus matrix order. For matrix* [1, 2, 1] *data points measured at matrix orders equal to multiples of* 32 *are connected with a solid line; others are joined with a dashed line.*

run of EISPACK's BISECT and TINVIT divided by the time for the parallel implementation on the slowest of the 32 processors. BISECT with TINVIT is the fastest method for finding all eigenvalues and eigenvectors of a symmetric, tridiagonal matrix of large order on one iPSC-1 processor, but the accuracy is not as high as that of Cuppen's method or TQL2. The eigensystem of the matrix [1, 2, 1] of order 512, for example, is computed in 2,340 seconds.

The speedup for [1, 2, 1] increases smoothly for matrix orders that are multiples of the number of processors. Efficiencies ranging from 77 percent to 89 percent are achieved for matrix orders greater than 100. Under the operating system in place on the iPSC-1 at the time of these experiments, a problem of order 3,520 is the largest that could be solved considering only orders that are multiples of 32. Comparable speedups found for random matrices of all orders show that the results are not strongly dependent on properties particular to matrix [1, 2, 1].

Efficient for other matrix orders fall to as much as 12 percent below the smooth line of those for multiples of 32. This reduction of speedup results from the fact that some processors compute one more eigenvalue and eigenvector than do others. The alternate direction exchange following the computation of eigenvalues (Step 2 of Algorithm B) synchronizes the processors so that those processors with a lesser workload are idle until the processors with a greater workload enter the exchange. Similarly, the eigenvector orthogonalization can be delayed by the uneven distribution of inverse iteration tasks. Hence, the completion time is determined by the processor with the largest work assignment. For example, for a matrix of order 65, one processor computes three eigenvalues while all others compute two. Because all processors wait for the processor computing three eigenvalues, the order 65 problem takes as long as if all processors were computing three eigenvalues (as would be the case for a matrix of order 96). The sequential solution of an order 65 problem, however, takes little more time than the solution of an order 64 problem but considerably less than an order 96 problem. Thus, the speedup for order 65 is appreciably less than that for order 64. In contrast, the speedup of an order 127 problem, for which both sequential and parallel times are close to those for order 128, is approximately equal to that of order 128.

Despite the extent of independent parallelism inherent in the bisection and inverse iteration procedures, maximal speedup is not achieved. The reduction in speedup can be attributed to nonarithmetic overhead as well as to some redundant (nonparallel) computations discussed below. Because the amount of communication in Algorithm B is dependent on the number and size of clusters in a matrix, the exact communication time cannot be measured by executing the algorithm without the arithmetic steps. Measurements of the nonarithmetic time thus contain time spent in waiting for messages as well as time spent in communication by a single processor. Figure 5 shows the average fraction of the total time spent idle or in communication by one processor. Again, the points for matrix [1, 2, 1] measured at orders divisible by 32 define a smooth curve falling from 20 percent of the total time at matrix order 32 to about 2 percent of the total for matrix orders larger than 320. An increase in nonarithmetic activity, due to the load imbalance discussed above, occurs for most orders not equal to multiples of 32. As expected, this increase is significant for order 65 but barely discernible for order 127.

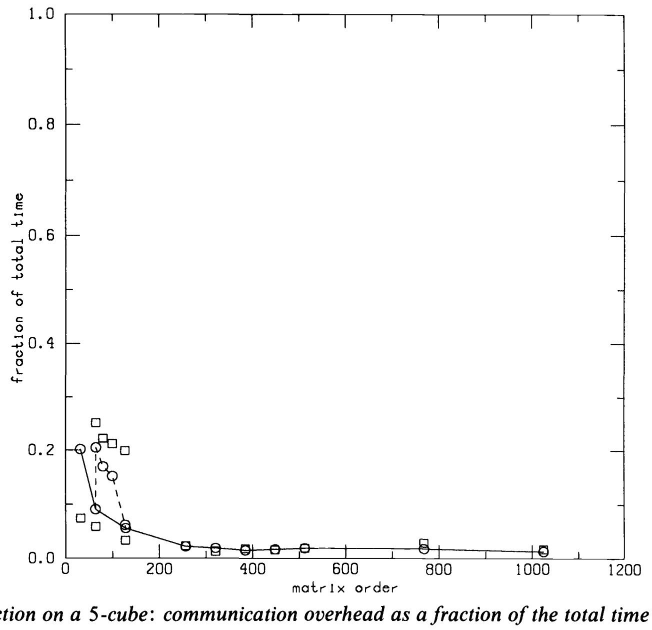


FIG. 5. *Bisection on a 5-cube: communication overhead as a fraction of the total time versus matrix order for matrix* [1, 2, 1].

The total time for Algorithm B is broken down into the times for the various steps as follows. The eigenvalue computation begins with each processor independently computing all Gerschgorin disks and subsequently carrying out an initial bisection to determine the interval containing its share of the eigenvalues. While this process could be altered to result in less redundancy, its present contribution to the total time is small. For a variety of matrices including [1, 2, 1], the initial bisection time decreases smoothly for all matrix orders from a maximum of about 15 percent of the eigenvalue computation time at order 32 to about 3 percent at order 512 and 1 percent at order 1,024. The grouping of close eigenvalues done in parallel by all processors occupies less than 5 percent of computation time for all matrix orders. The time needed for starting the pipeline of communication for Algorithm MGS represents an additional loss of efficiency amounting to less than 2 percent of the total time for all orders of matrix [1, 2, 1]. Thus, at order 32, about 42 percent of the total time is spent in communication and redundant arithmetic. (From the previous remarks on nonarithmetic overhead, 22 percent of the time can thus be attributed to redundant arithmetic.) The 42 percent overhead time completely accounts for the observed efficiency of 58 percent. Similarly, the 10 percent of the total time spent in overhead for order 1,024 approximates the observed lowering of the speedup curve below optimal.

Figure 6 shows the average fraction of the total time spent in computing, exchanging, and grouping eigenvalues, in determining eigenvectors, and in orthogonalizing eigenvectors corresponding to clusters for several orders of matrix [1, 2, 1]. Finding and distributing the eigenvalues to all processors occupies more than 80 percent of the total time, while computing the eigenvectors occupies most of the remaining time. The low contribution of the orthogonalization step confirms the effectiveness of the cyclic distribution approach for the modified Gram-Schmidt procedure. In contrast, modified Gram-Schmidt occupies 2.6 percent of the total time for order 500 and 10.2 percent of the total for order 1,000 for a similar algorithm executed on an Alliant FX/8 [17]. These time distributions are particular to the matrix [1, 2, 1]. When eigenvalues are more strongly clustered, bisection generally occupies a smaller fraction of the time while reorthogonalization requires more.

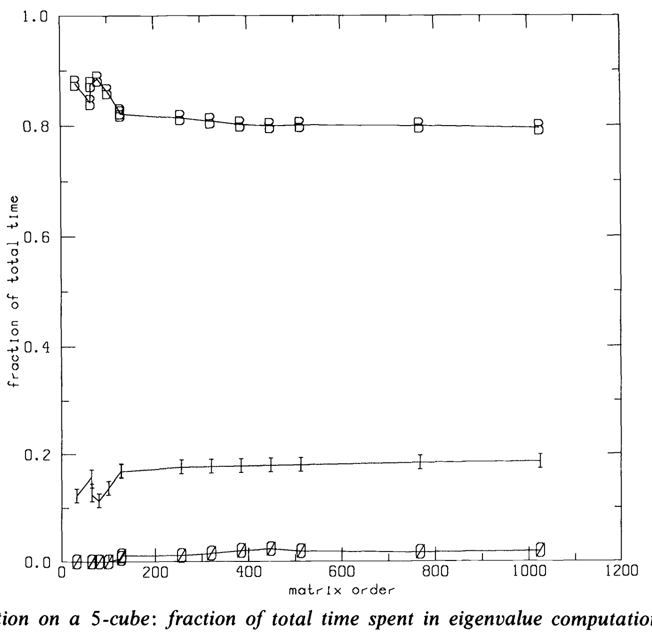


FIG. 6. *Bisection on a* 5-*cube*: *fraction of total time spent in eigenvalue computation* (B), *eigenvector computation* (I), *and orthogonalization* (O) *versus matrix order.*

As for the speedup and communication curves, data points in Fig. 6 for matrix orders that are multiples of 32 lie on a smooth curve, and points for other orders do not. The increase in eigenvalue computation time for the deviant points corresponds to the increase in nonarithmetic time (see Fig. 5). Again, the results are problem dependent. The distribution of eigenvalues of matrix [1, 2, 1] leads to a fairly even distribution of work among processors. Certain spectra could lead to load imbalance especially during bisection or reorthogonalization but were not observed in a variety of random and other test matrices.

5.3. Analysis of a model problem. While the exact time contribution of each computing task is problem dependent, analytic examination of a simple model problem can shed light on the arithmetic requirements of the bisection method. Consider the symmetric, tridiagonal matrix T_{model} of order $N = kp$ having eigenvalues $0, \alpha/N, \cdots, (N-1)\alpha/N$. Suppose that the eigenvalues are known a priori to be confined to the interval $[0, \alpha)$, then the spectrum of T_{model} approximates that of [1, 2, 1], which has N eigenvalues initially confined within an interval of length four for all values of N.

During the initial bisection of Algorithm B, each processor finds an interval of length α/p containing k eigenvalues. This step takes $l \geqq \log_2 p$ iterations for a total of $l+1$ Sturm sequence evaluations. In subsequent steps, each processor extracts its k eigenvalues. Finding its largest eigenvalue first, each processor reduces an interval of width α/p to one of width δ in $l_1 \geqq \log_2 \alpha/p\delta$ bisections. After this computation, the portion of the interval following the first eigenvalue computed is discarded, leaving a new search interval of length $\alpha/p - \alpha/N$. The second eigenvalue is then extracted in $l_2 \geqq \log_2 (\alpha/p - \alpha/N)/\delta$ bisections. In general, the ith eigenvalue is found in an interval of width $\alpha/p-(i-1)\alpha/N$ in $l_i \geqq \log_2 (\alpha/p-(i-1)\alpha/N)/\delta$ bisections, a process requiring $l_i + 1$ Sturm sequence evaluations. If the time to complete one Sturm sequence evaluation is $2N\omega$, the total time for the eigenvalue computation is

$$T_B \geqq \left[(l+1) + \sum_{i=1}^{k} \left(\log_2 \frac{\alpha i}{N\delta} + 1\right)\right] 2N\omega$$

$$\approx \left[\log_2 p + 1 + k \log_2 \frac{\alpha k}{N\delta} + k\right] 2N\omega.$$

The eigenvectors are computed using inverse iteration. For a tridiagonal matrix, each iteration requires time of about $5N\omega$. Assuming that convergence is achieved in an average of two iterations, computing k eigenvectors takes time $T_I = 10Nk\omega$. The spacing of the eigenvalues is assumed wide enough that additional orthogonalization of eigenvectors is not required.

The attainable tolerance δ is related to both machine precision and the largest eigenvalue magnitude [22]. For the model problem in double precision, $\delta \approx 10^{-15}\alpha \approx 2^{-50}\alpha$, and T_B/T_I decreases from 12 to about 11 when the matrix order increases from 32 to 1,024 on a 5-cube. Thus, the eigenvalue computation dominates the total arithmetic time for the model problem. The eigenvalues of matrix [1, 2, 1] are more closely spaced than those of T_{model} and so require fewer bisections to extract. For this reason, the eigenvalue computation for [1, 2, 1] occupies about four times the time of its eigenvector computation. (As indicated by Fig. 5, nonarithmetic operations contribute minimally to the experimental result.) The predicted reduction in eigenvalue computation time with respect to eigenvector time for increasing matrix order of T_{model} is also reflected in Fig. 6. For [1, 2, 1], T_B/T_I falls from 7.1 at order 32 to 4.3 at order 1,024.

The model problem further follows the experimental results by having a communication complexity significantly smaller than its arithmetic complexity and decreasing with matrix order. A single alternate direction exchange of N/p eigenvalues per processor comprises the total communication requirement of Algorithm B. Assuming $\beta \approx 10\omega$ and $\tau \approx 10\omega/125$, the communication time comes to

$$T_C = 2\left(\beta \log_2 p + (p-1)\frac{N}{p}\tau\right)$$
$$\approx 20\left(\log_2 p + \frac{p-1}{125p}N\right)\omega.$$

For the model problem on a 5-cube, the ratio of communication time to computation time $T_C/(T_B+T_I)$ falls from .02 at order 32 to 3×10^{-5} at 1,024. The percentage of communication time for $[1,2,1]$ in Fig. 5 is greater than that of T_{model} because the eigenvalue computation in T_{model} takes longer. As shown in Fig. 6, orthogonalization does not greatly affect the run time for matrix $[1,2,1]$.

6. Multisection and inverse iteration. Bisection involves repeated halving of a search interval. The generalization of bisection known as *multisection* is the division of an interval into $p \geqq 2$ equal-size subintervals; Sturm sequence evaluations are used to find the number of eigenvalues in each of the p subintervals. Subintervals found to contain more than one eigenvalue are in turn multisected in order to locate individual or clustered eigenvalues. As in bisection, empty subintervals are immediately discarded. Once an interval contains a single eigenvalue, multisectioning is discontinued regardless of the interval width. In the terminology of [17], an eigenvalue confined alone in this way has been *isolated.* Single eigenvalues determined to within a given tolerance δ have been *extracted.* Several eigenvalues contained in an interval of width less than δ are not further isolated but rather form an *extracted cluster.* In the implementation described below, δ is taken to be the small multiple of machine epsilon used in EISPACK's BISECT as the criterion for accepting computed eigenvalues.

As shown in [17], it is faster on one processor to extract a single, isolated eigenvalue by bisection rather than multisection. Even faster interpolation methods are not considered for extraction due to the possibility of under- and overflow mentioned in § 5. Moreover, using interpolation in the parallel implementations of bisection and multisection is not likely to affect their speeds with respect to each other and would not affect the conclusions drawn in § 7.

The procedure for finding all eigenvalues of a symmetric, tridiagonal matrix using multisection and the corresponding eigenvectors with inverse iteration is summarized in Algorithm M1.

ALGORITHM M1. Multisection and inverse iteration.

Step 0. *Determination of initial interval.* Compute all Gerschgorin disks to provide an interval containing all eigenvalues.

Step 1. *Isolation of eigenvalues.* Use multisection recursively to divide intervals until each interval holds either one eigenvalue or an extracted cluster of eigenvalues.

Step 2. *Extraction of eigenvalues.* Recursively use bisection to divide intervals containing one eigenvalue until the eigenvalue has been extracted to a specified tolerance δ.

Step 3. *Computation of eigenvectors.* Perturb clustered eigenvalues, and compute all eigenvectors by inverse iteration.

Step 4. *Orthogonalization of eigenvectors.* Orthogonalize eigenvectors corresponding to clusters.

6.1. Parallel algorithm. In Algorithm B, bisection of an interval is carried out by a single processor. Parallel multisection on a cube of $p = 2^d$ processors provides a means of using all p processors to find the eigenvalues in one interval. The parallel implementation of multisection is based on Algorithm M1. Computation of eigenvalues begins with *each* processor finding the initial interval containing all eigenvalues (e.g., by computing the Gerschgorin disks). The computing tasks for isolation and extraction of eigenvalues are shared by all processors; eigenvector determination proceeds as in Algorithm B. The approaches to load balancing taken during eigenvalue computation are as follows.

(1) *Isolation of eigenvalues.* To begin, the initial interval is divided into p equal-size subintervals, and processor i determines the number of eigenvalues located in the ith subinterval. All processors then employ Algorithm ADE to exchange their eigenvalue counts. Using the collected numbers of eigenvalues in the p intervals, each processor then discards empty intervals and retains in the *extraction list* intervals containing a single, isolated eigenvalue. Each processor places in the *isolation queue* all intervals requiring further parallel multisectioning. In lock step, the processors perform multisection on the interval currently at the head of the isolation queue; processor i counts the eigenvalues in the resulting ith subinterval.

An interval containing more than one eigenvalue is not further subjected to multisection when it is too small to be divided into p subintervals, i.e., when its width is less than $p\delta$. Because its width is not necessarily less than δ, the interval is entered into the extraction list for further processing. Thus, every processor creates and maintains an identical queue of remaining multisection tasks and a list of extraction tasks.

Multisectioning continues until the isolation queue is empty, that is, until all single eigenvalues have been isolated and the intervals with several eigenvalues are too small to be divided into p subintervals.

(2) *Extraction of eigenvalues.* The sets of intervals stored in the extraction list during the isolation phase define two distinct sets of tasks, where each such task is to be executed on one processor. A task is either the extracting of an isolated single eigenvalue from an interval of arbitrary length via bisection or the multisecting of an interval of length less than $p\delta$ containing several eigenvalues into subintervals of size δ.

The simple synchronization scheme from the isolation phase is no longer applicable. An alternative approach to load balancing must be considered. A dynamic queue management scheme can be inefficient on the hypercube. Moreover, the problem of statically scheduling a set of nonidentical tasks so as to minimize completion time is NP-complete [21]. However, among standard weighted scheduling algorithms, the Longest Job First (LJF) schedule provides near optimal performance [13]. In this heuristic, the most time consuming remaining job is given to the processor with the lightest total workload.

As it maintains the extraction list, each processor in the hypercube can estimate the time required to extract the eigenvalues from each interval. Extraction of an isolated eigenvalue in an interval of size I by bisection requires $\lceil \log_2 I/\delta \rceil$ iterations for a total of $\lceil \log_2 I/\delta \rceil + 1$ Sturm sequence evaluations. Multisectioning an interval of size $I < p\delta$ with several eigenvalues into $I/\delta < p$ subintervals can be completed in a single iteration with no more than $I/\delta + 1$ Sturm sequence evaluations. A rough estimate of the time required to extract the eigenvalues from an interval can be based solely on the size of the interval: every extraction task is assigned a cost of $\lceil \log_2 I/\delta \rceil + 1$ if it requires bisection, and $I/\delta + 1$ if it requires multisection. The LJF implementation makes use of a heap to record task assignments for all processors. As straightforward manipulation of a heap constitutes a sequential process, the scheduling procedure is performed by every processor. After all eigenvalues and clusters of eigenvalues have been extracted, an alternate direction exchange ensures that all processors have all eigenvalues.

Multisection on the hypercube is summarized as Algorithm M2.

ALGORITHM M2. Solution of eigenproblem of order $N = kp$ on a p-processor hypercube. Each processor contains all diagonal and off-diagonal elements of the matrix. In parallel, all p processors perform the following steps.

Step 0. *Determination of initial interval.* All processors compute all Gerschgorin disks to provide an initial interval containing all eigenvalues. Isolated disks are added to the extraction list and unions of overlapping disks to the isolation queue.

Step 1. *Isolation of eigenvalues.* Each processor divides the first interval in the isolation queue into p equal-size subintervals, and processor i determines the number of eigenvalues in the ith subinterval. All p processors use Algorithm ADE to exchange their eigenvalue counts. Each processor discards empty subintervals and adds to the extraction list subintervals with one eigenvalue and subintervals of size less than $p\delta$ with several eigenvalues. It appends the remaining subintervals (of size $p\delta$ or larger) with several eigenvalues to the isolation queue. Step (1) is repeated until the isolation queue is empty.

Step 2. *Extraction of eigenvalues.* Each processor determines its share of eigenvalues from the extraction list according to a LJF scheduling heuristic. Each processor bisects intervals with single eigenvalues, and multisects intervals with several eigenvalues.

Step 3. *Exchange of eigenvalues.* All processors exchange their eigenvalues by means of Algorithm ADE.

Step 4. *Perturbation of eigenvalues.* Each processor perturbs clustered eigenvalues.

Step 5. *Computation of eigenvectors.* Processor i computes N/p eigenvectors corresponding to eigenvalues indexed $i, i+p, \cdots, i+\nu p \leqq N$ with inverse iteration.

Step 6. *Orthogonalization of eigenvectors.* Each processor orthogonalizes those of its eigenvectors associated with a cluster via Algorithm MGS.

6.2. Experimental results. As in bisection, the speedup for multisection is calculated as the time for EISPACK's BISECT and TINVIT combination run on a single processor divided by the time for the parallel implementation on the slowest of the iPSC-1/d5M processors. The results are given in Fig. 7.

The speedup for matrix $[1, 2, 1]$ increases monotonically from 8.5 at order 64 to 19.4 at order 1,024. The slight increase in speedup at order 32 over that at order 64 is an artifact of the problem size equaling the number of processors. Between orders 64 and 1,024, the efficiency ranges between 26 percent and 60 percent as compared to 74 percent to 89 percent for bisection applied to the same set of problems (see Fig. 4). The lower efficiency of multisection is due, in part, to increased overhead. Figure 8 shows that the average communication and idle time account for over 40 percent of the total time at low orders and for 5 percent of the total time at order 1,024. In contrast, overhead for bisection applied to the same set of problems varies from 20 percent to 2 percent of the total time (see Fig. 5). Our implementation of multisection exhibits redundant arithmetic in the perturbation of clustered eigenvalues by all processors, in the pipeline startup time for orthogonalization, and in the load balancing

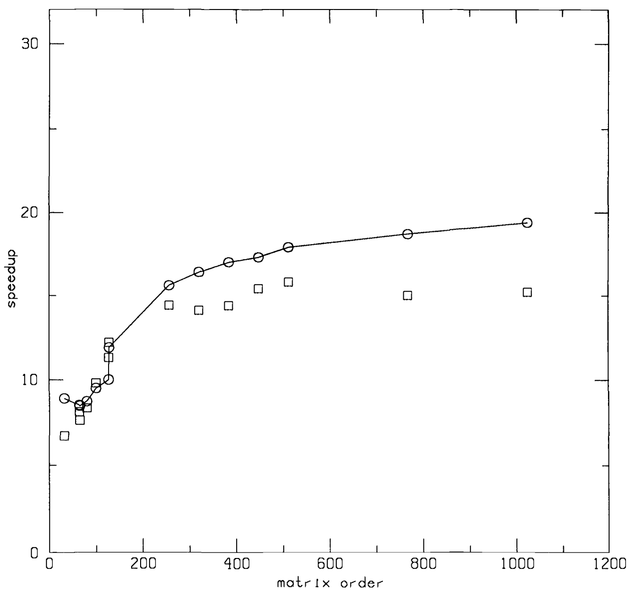


FIG. 7. *Multisection on a 5-cube: speedup on the* iPSC-1 *for* $[1,2,1]$ *(circles) and random matrices (squares) versus matrix order.*

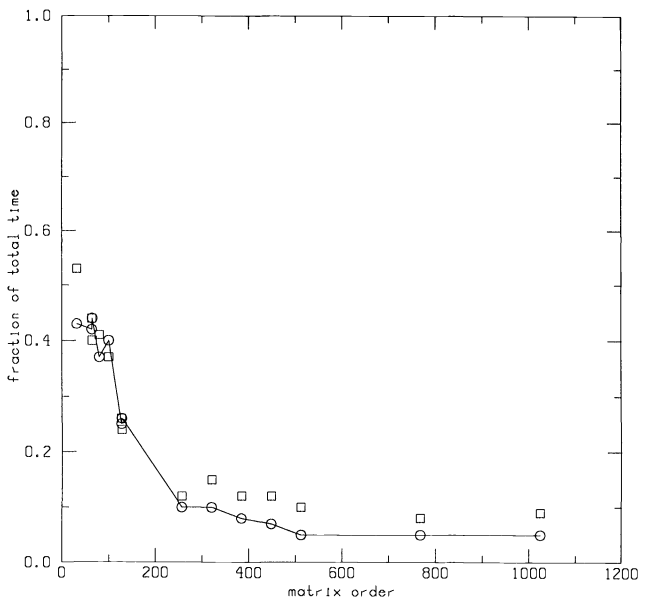


FIG. 8. *Multisection on a 5-cube: communication overhead as a fraction of the total time versus matrix order.*

routines. This accounts for a combined total of less than 12 percent of the computation time for all matrix orders. The remaining loss of speedup can be attributed to inefficiency in the actual isolation and extraction processes.

The average fractions of the total time spent in computation and perturbation of eigenvalues, in computation of eigenvectors, and in orthogonalization of eigenvectors are given in Fig. 9. Eigenvalue computations take more than 90 percent of the total

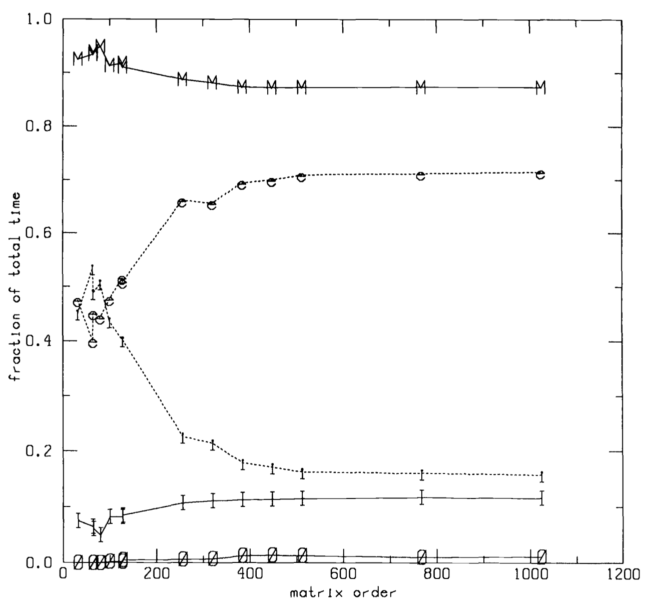


FIG. 9. *Multisection on a* 5*-cube*: *fraction of total time spent in eigenvalue computation* (M), *eigenvector computation* (I), *and orthogonalization* (O) *versus matrix order. The fractions of time spent in isolation* (i) *and extraction* (e) *sum to the fraction in eigenvalue computation.*

time for all matrix orders; eigenvector computations grow from 7.5 percent to 11.6 percent of the total for orders increasing from 32 to 1,024, while orthogonalization occupies up to 1.4 percent of the total. The steps of Algorithms M2 and B pertaining to eigenvector computation and orthogonalization are identical and therefore take equal time. As these computations represent 1.9 percent-18.6 percent of the total time for bisection but only 1.4 percent-11.6 percent of the time for multisection, multisection contains a larger contribution of eigenvalue computations and exhibits a greater elapsed time. Some sample times for bisection and multisection given in § 7 confirm that bisection is indeed the faster of the two.

6.3. The model problem revisited. A return to the model problem of § 5.3 helps to explain Fig. 9. Recall that the tridiagonal matrix T_{model} of order $N = kp$ has eigenvalues $0, \alpha/N, \cdots, (N-1)\alpha/N$, and the initial interval is known to be $[0, \alpha)$.

Algorithm M2 begins by isolating the eigenvalues with multisection. The initial interval of length α is divided into p intervals of length α/p, each containing k eigenvalues. Each of the p intervals is in turn multisected to create a total of p^2 intervals, each containing k/p eigenvalues. In general, the ith multisection step produces p^i intervals containing N/p^i eigenvalues apiece. For the model problem, multisectioning stops once each interval contains at most one eigenvalue, that is, after j steps, where $p^j \leqq N < p^{j+1}$. Since an interval contains no more than one eigenvalue, its length is at most α/N.

During a multisection step, a processor must evaluate Sturm sequences for both endpoints of each of its intervals. The total number of Sturm sequence evaluations for the isolation phase is then

$$S_{isol} = 2 \sum_{i=0}^{j} p^i \geqq 2\frac{N-1}{p-1}.$$

Each Sturm sequence evaluation takes time $2N\omega$ so that

$$T_{isol} \geqq 4\frac{N-1}{p-1}N\omega.$$

Isolation leaves in every processor a total of $p^j \leqq N$ intervals of length at most α/N containing one eigenvalue. Extracting a single eigenvalue to tolerance δ from an interval requires l bisections, where $2^l \geqq \alpha/N\delta$. The number of Sturm sequence evaluations performed by one processor during extraction is then

$$S_{extr} = k(l+1) \geqq k\left(\log_2 \frac{\alpha}{N\delta}+1\right),$$

as $p^j \geqq N/p = k$. The total time for extraction is

$$T_{extr} \geqq k\left(\log_2 \frac{\alpha}{N\delta}+1\right)2N\omega.$$

For all orders N, the union of Gerschgorin disks comprising the initial interval of matrix $[1,2,1]$ covers the interval $[0,4)$. Thus, with increasing order, an increasing number of eigenvalues is determined from an interval of constant length. Because its eigenvalues are confined to an interval of size α, the matrix T_{model} can thus be used to approximate the multisection time for matrix $[1,2,1]$.

Because the eigenvectors are distributed cyclically, each processor in Algorithm B determines k eigenvectors of T_{model} by inverse iteration. This computation takes time $T_I = 10\,Nk\omega$. The distance α/N between adjacent eigenvalues is assumed large enough that none of the eigenvectors need be orthogonalized, so the ratio of eigenvalue to eigenvector computation times for T_{model} is about

$$\frac{T_M}{T_I} = \frac{T_{isol}+T_{extr}}{T_I} \approx \frac{1}{5k}\left(2\frac{N-1}{p-1}+k\log_2\frac{\alpha}{N\delta}+k\right).$$

T_M/T_I falls from about 10 at matrix order 32 to about 9 at order 1,024; these values are in close agreement with those obtained experimentally for matrix $[1,2,1]$.

For matrix orders below 200 in Fig. 9, extraction and isolation take approximately equal portions of the total time. As the order increases, extraction dominates. For the model problem, the number of Sturm sequence evaluations in isolation differs from that in extraction by

$$D = S_{extr} - S_{isol} = k\left(\log_2\frac{\alpha}{N\delta}+1\right) - 2\frac{N-1}{p-1}.$$

D is positive whenever $\alpha/N \geqq \delta$, meaning that more Sturm sequence evaluations are performed in extracting the eigenvalues of T_{model} than in isolating them. The derivative of D with respect to k is

$$\frac{dD}{dk} = \log_2\frac{\alpha}{N\delta}+1-\log_2 e - 2\frac{p}{p-1} \approx \log_2\frac{\alpha}{N\delta}.$$

This quantity is greater than zero when $\alpha/N > \delta$ and decreases in magnitude with increasing N. Hence, D for the model problem has the same qualitative behavior as recorded for matrix $[1,2,1]$.

While the model problem helps to explain some of the experimental results for $[1,2,1]$, it does not account for the fact that the observed speedup of multisection is less than that of bisection for all orders of matrix $[1,2,1]$. For T_{model}, $T_B > T_M$ for all matrix orders, meaning that the predicted speedup is in fact greater for multisection than for bisection. This discrepancy stems from the differences in the spectra of the two matrices. While the eigenvalues of T_{model} are uniformly spaced, the eigenvalues of the Toeplitz matrix $[1, 2, 1]$ of order N are given by $\lambda_i = 2(1+\cos(i\pi/(N+1)))$ for $i = 0, 1, \cdots, N-1$ [14] and so are more tightly spaced at the extremes of the spectrum than near the center. Isolating a set of close eigenvalues takes longer than isolating an equally large set of well-separated eigenvalues, and the former eigenvalues are confined to smaller intervals than the latter. Thus, once isolated, close eigenvalues are extracted faster than well separated ones.

An extreme example shows that the relative numbers of Sturm sequence evaluations for multisection and bisection are strongly dependent not only on the number of eigenvalues in the initial search area but also on their distribution within that interval. Let T'_{model} be a matrix of order N having $N = kp$ eigenvalues of multiplicity 2 uniformly distributed at spacing $2(\alpha/N)$ in the interval $[0, \alpha)$. Consider the case $N = 2p$. The number of Sturm sequence evaluations needed to determine all eigenvalues to a tolerance δ via Algorithm B on a p-processor hypercube is

$$S'_B = \log_2 \frac{\alpha}{\delta} + 2.$$

The number needed to find the eigenvalues using Algorithm M2 is

$$S'_M = 2\frac{1}{\log_2 p}\log_2 \frac{\alpha}{\delta} + p - 3 \quad \text{if } p^k(p-1)\delta \leqq \alpha \leqq p^{k+1}\delta \quad \text{for some } k \geqq 1.$$

Hence, $S'_M > S'_B$ if $k \lesssim p/\log_2 k$, and the speedup of bisection is greater than that of multisection.

It is evident from Fig. 9 that the speedup worsens when isolation dominates. For matrix order $N = 32$, $T_{isol}/T_{extr} \approx 1.3$, and the speedup of multisection is 35 percent that of bisection; while for $N = 1{,}024$, $T_{isol}/T_{extr} \approx .27$, and the speedup for multisection is 67 percent that of bisection. The times and speedups for random matrices of order $N = 64$ are the same as for $[1, 2, 1]$, and for $N = 1{,}024$, $T_{isol}/T_{extr} \approx .46$ and the ratio of speedups is .54. For intermediate orders of $[1, 2, 1]$ and random matrices, increased isolation also leads to decreased speedup.

Matrices $[1, 2, 1]$, T_{model}, and T'_{model} all indicate the slowdown of multisection as eigenvalues become increasingly clustered. Attempts to generalize this information by classifying matrices according to size and number of eigenvalue clusters were unsuccessful, as other problem dependent factors conceal any structure.

7. Comparison and conclusions. Tables 2 and 3 show the total time, the residual, and the deviation from orthogonality for several orders of matrix $[1, 2, 1]$ and of random matrices for all three eigensolvers. Data communication on the hypercube does not alter the numerical properties of the methods. Hence Cuppen's method gives the most accurate results, consistently yielding smaller residuals and deviations from orthogonality than both bisection and multisection for all tested matrices.

Bisection is the fastest method for finding all the eigenvalues and eigenvectors at all orders. Although for at least one problem multisection is theoretically faster than bisection, in practice, multisection is slower due to arithmetic inefficiency in the isolation phase. This implementation of multisection and Cuppen's method are of comparable speed at low orders, while for larger orders multisection is the faster of the two. The speedups of Cuppen's method, bisection, and multisection over the fastest sequential method are problem dependent. The figures for the tested matrices suggest that maximal speedup cannot be expected for any of the three methods. Speedup of Cuppen's method

TABLE 2
Comparison of methods for matrix [1, 2, 1] *on a* 5*-cube.*

Method	Order	Time (seconds)	Residual $\|TX-\Lambda X\|$	Orthogonality $\|X^TX-I\|$
Cuppen's	32	1.3	9.2e−16	7.0e−16
Bisection		0.6	9.4e−15	3.3e−14
Multisection		1.2	7.4e−15	3.2e−14
Cuppen's	100	10.5	1.9e−15	1.9e−15
Bisection		4.5	3.3e−14	3.1e−14
Multisection		9.7	1.3e−14	2.8e−14
Cuppen's	512	611.8	8.4e−15	1.8e−14
Bisection		88.7	8.8e−13	6.0e−13
Multisection		141.5	6.1e−13	3.9e−13

TABLE 3
Comparison of methods for random matrices on a 5*-cube.*

Method	Order	Time (seconds)	Residual $\|TX-\Lambda X\|$	Orthogonality $\|X^TX-I\|$
Cuppen's	32	1.1	2.7e−15	2.7e−15
Bisection		0.5	2.9e−15	3.0e−14
Multisection		1.6	5.9e−15	1.2e−13
Cuppen's	100	10.4	7.4e−15	8.9e−15
Bisection		4.6	3.6e−14	6.5e−14
Multisection		10.7	4.7e−14	7.1e−14
Cuppen's	512	623.9	7.9e−15	1.3e−14
Bisection		88.3	5.3e−13	2.1e−13
Multisection		160.3	6.5e−13	7.1e−12

is especially reduced when significant deflation occurs in some but not all subproblems. Note that all experimental results are problem dependent, although the conclusions drawn here hold for a large number of tested matrices. A closer examination of the effects of eigenvalue clustering on speed and accuracy will be included in a future paper.

Although not discussed in this paper, the sectioning algorithms are readily modified to permit computation of a subset of eigenvalues and eigenvectors. Cuppen's method can only be recommended when the entire eigensystem is needed. In that event, Cuppen's method, which requires sufficient storage for parallel matrix multiplication by Algorithm RMM, can be used for problems only as large as order 2,080 on the iPSC-1/d5M. Bisection and multisection, which require no matrix multiplication, can both be used for problems through order 3,520.

Comparison with shared-memory implementations. The efficiency of solving the symmetric, tridiagonal eigenproblem on the local-memory hypercube as opposed to a shared-memory multiprocessor is dependent on the method used. In [17], timings are presented for SESUPD (a shared-memory implementation of Cuppen's method [8]), BISECT with TINVIT, and TREPS1 (multisection using bisection during the extraction phase) on eight processors of an Alliant FX/8. In particular, speedups of these algorithms with respect to the time to find all eigenvalues and eigenvectors of matrix $[-1, 2, -1]$ of order 500 via EISPACK's TQL2 on one processor are given. The efficiency of algorithm i compared to TQL2 can be defined by

$$E_i = \frac{\text{time for TQL2 on 1 processor}}{p * (\text{time for algorithm } i \text{ on } p \text{ processors})}.$$

The shared-memory implementations then have $E_{\text{SESUPD}} = 3.4$, $E_{\text{BISECT}} = 0.5$, and $E_{\text{TREPS1}} = 4.0$. In contrast, the equivalent algorithms for the 5-cube have values $E_{C2} = 1.4$, $E_B = 10.0$, and $E_{M2} = 6.2$ for matrix $[1, 2, 1]$ of order 512. These relative efficiencies are greater than unity because TQL2 is slower on one processor than either SESUPD or BISECT. The greatest values of E_i occur for the algorithms that are most effectively implemented in parallel. The independent tasks of the bisection and multisection algorithms are especially adaptable to a local-memory architecture. Cuppen's method, on the other hand, is seen to be more efficient on the shared-memory machine, where a dynamic scheduler is supplied.

The extensive communication requirements of a dynamic scheduler on a local-memory machine and the potential loss of computing power from processors dedicated to scheduling tasks recommend the use of static scheduling schemes. The resulting loss of parallelism at the root finding level for Cuppen's method and the simultaneous execution of an explicit scheduler on all processors for multisection, however, prevent full speedup.

Acknowledgments. The authors thank Stan Eisenstat, Bill Gropp, Cleve Moler, and Dan Sorensen for many helpful discussions.

REFERENCES

[1] R. H. BARLOW AND D. J. EVANS, *A parallel organization of the bisection algorithm*, Comput. J., 22 (1977), pp. 267-269.

[2] R. H. BARLOW, D. J. EVANS, AND J. SHANEHCHI, *Parallel multisection applied to the eigenvalue problem*, Comput. J., 26 (1983), pp. 6-9.

[3] H. J. BERNSTEIN, *An accelerated bisection method for the calculation of eigenvalues of a symmetric tridiagonal matrix*, Numer. Math., 43 (1984), pp. 153-160.

[4] H. J. BERNSTEIN AND M. GOLDSTEIN, *Parallel implementation of bisection for the calculation of eigenvalues of tridiagonal symmetric matrices*, Computing, 37 (1986), pp. 85-91.

[5] H. BOWDLER, R. S. MARTIN, AND J. H. WILKINSON, *The* QR *and* QL *algorithms for symmetric matrices*, Numer. Math., 11 (1968), pp. 227-240.

[6] J. R. BUNCH, C. P. NIELSEN, AND D. C. SORENSEN, *Rank-one modification of the symmetric eigenproblem*, Numer. Math., 31 (1978), pp. 31-48.

[7] J. J. M. CUPPEN, *A divide-and-conquer method for the symmetric tridiagonal eigenproblem*, Numer. Math., 36 (1981), pp. 177-195.

[8] 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.

[9] G. E. FORSYTHE, M. A. MALCOLM, AND C. B. MOLER, *Computer Methods for Mathematical Computations*, Prentice Hall, Englewood Cliffs, NJ, 1977.

[10] G. FOX, A. J. G. HEY, AND S. OTTO, *Matrix algorithms on the hypercube* I: *matrix multiplication*, Tech. Report, California Institute of Technology, Pasadena, CA, 1985.

[11] E. N. GILBERT, *Gray codes and paths on the N-cube*, Bell Sys. Tech. J., 37 (1958), pp. 815-826.

[12] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.

[13] R. L. GRAHAM, *Bounds on multiprocessor timing anomalies*, SIAM J. Appl. Math., 17 (1969), pp. 416-429.

[14] R. T. GREGORY AND D. L. KARNEY, *A Collection of Matrices for Testing Computational Algorithms*, John Wiley, New York, 1969.

[15] I. C. F. IPSEN AND E. R. JESSUP, *Solving the symmetric tridiagonal eigenvalue problem on the hypercube*, Res. Report 548, Department of Computer Science, Yale University, New Haven, CT, 1987.

[16] A. S. KRISHNAKUMAR AND M. MORF, *Eigenvalues of a symmetric tridiagonal matrix: a divide-and-conquer approach*, Numer. Math., 48 (1986), pp. 349-368.

[17] S. LO, B. PHILLIPE, AND A. SAMEH, *A multiprocessor algorithm for the symmetric tridiagonal eigenvalue problem*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s155-s165.

[18] E. M. REINGOLD, J. NIEVERGELT, AND N. DEO, *Combinatorial Algorithms*, Prentice Hall, Englewood Cliffs, NJ, 1977.

[19] Y. SAAD AND M. H. SCHULTZ, *Data communication in hypercubes*, Res. Report 428, Department of Computer Science, Yale University, New Haven, CT, 1985.

[20] ———, *Some topological properties of the hypercube multiprocessor*, Res. Report 389, Department of Computer Science, Yale University, New Haven, CT, 1984.

[21] S. K. SAHNI, *Algorithms for scheduling independent tasks*, Assoc. Comput. Mach., 23 (1976), pp. 116-127.

[22] B. T. SMITH, J. M. BOYLE, J. J. DONGARRA, B. S. GARBOW, Y. IKEBE, V. C. KLEMA, AND C. B. MOLER, *Matrix Eigensystem Routines*—EISPACK *Guide*, Second edition, Lecture Notes in Computer Science, 6, Springer-Verlag, Berlin, New York, 1976.

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

[24] A. Y. WU, *Embedding of tree networks into hypercubes*, J. Par. Distr. Comput., 2 (1985), pp. 238-249.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 2, pp. 230-241, March 1990

COMPUTATION OF GEODESIC TRAJECTORIES ON TUBULAR SURFACES*

JAMES SNEYD† AND CHARLES S. PESKIN†

Abstract. This paper describes a method for computing geodesics on an arbitrary tubular surface. Such a surface is defined by specifying a centreline curve and a radius function. The equations of a geodesic are formulated in terms of these given data, and the resulting system of ordinary differential equations is solved by a second-order Runge-Kutta method. Computed results reveal the phenomenon of geodesic confinement in a region containing a bulge of the tube. Numerical evidence of convergence is presented, and possible applications to the study of blood flow in fibre-reinforced vessels and in the developing heart are briefly discussed.

Key words. computation, fibres, geodesics, heart

AMS(MOS) subject classifications. 34A34, 34A50, 65L05, 92-04, 92A05

1. Introduction.

This paper is concerned with the computation of geodesic paths on a class of surfaces that we call *tubular.* The motivation for this work is the need to model elastic blood vessel walls for computations involving blood flow in such a vessel. Blood vessel walls are fibre-reinforced, and it is a reasonable hypothesis that the fibres follow geodesic paths. Therefore, to construct a model of a blood vessel, we need a method for wrapping computational fibres around surfaces that lie in the interior of the vessel walls.

For purposes of this paper, a *tubular* surface $\mathcal{S}$ is defined by specifying a centreline curve $\mathcal{C}$ in parametric form, $\mathbf{X}=\mathbf{X}_0(t)$ as well as a radius function $R(t)$. Although it is not necessary for $\mathcal{C}$ to be parameterised by arclength, this assumption will simplify the following discussion; therefore we assume that t is arclength along $\mathcal{C}$. For fixed t, consider the circle $\mathcal{A}(t)$ given by

$$|\mathbf{X}-\mathbf{X}_0(t)|=R(t), \qquad (\mathbf{X}-\mathbf{X}_0(t))\cdot\frac{d\mathbf{X}_0(t)}{dt}=0.$$

Note that $d\mathbf{X}_0(t)/dt$ is the unit tangent to $\mathcal{C}$ at $\mathbf{X}_0(t)$. The tubular surface $\mathcal{S}$ is defined as the surface swept out by the circle $\mathcal{A}$ as t varies; to simplify the problem, we insist that $\mathcal{S}$ does not intersect itself and that it has no folds. Note that $\mathcal{S}$ can also be defined as the set of all points $\mathbf{X}$ that satisfy

$$f(\mathbf{X},t)=\frac{1}{2}\{|\mathbf{X}-\mathbf{X}_0(t)|^2-R(t)^2\}=0,$$

$$g(\mathbf{X},t)=(\mathbf{X}-\mathbf{X}_0(t))\cdot\frac{d\mathbf{X}_0(t)}{dt}=0$$

for any t. We are therefore led to consider the general problem of finding geodesics on surfaces defined by equations of the form $f(\mathbf{X},t)=0$, $g(\mathbf{X},t)=0$.

An important feature of the method that we develop is that we do not use t as the independent variable along the geodesic path. Rather, we solve for t as well as $\mathbf{X}$

* Received by the editors March 7, 1988; accepted for publication (in revised form) December 21, 1988. This work was supported by the National Institutes of Health under grant HL17859 and by the National Science Foundation under grant DMS8701895. Computation was performed at the Courant Institute of Mathematical Sciences Computing Laboratory, New York University, New York, 10012 under Department of Energy contract DEAC0276ER03077.

† Courant Institute of Mathematical Sciences, 251 Mercer Street, New York, New York 10012.

as functions of arclength s on the geodesic. This allows for the case in which the geodesic "turns around," i.e., in which dt/ds changes sign. This sometimes happens to geodesics on tapering surfaces. In fact, as we shall see, the geodesic may become "trapped" in a region where the tube has a bulge; in such a case it will cycle back and forth between two extreme values of t.

2. Determination of geodesic equations for a general surface. Given a surface $F(\mathbf{X})=0$ embedded in three-space we can define a geodesic on the surface as a curve whose principal normal is parallel to $\nabla F(\mathbf{X})$. The equations for such a geodesic curve can be found in any standard text on differential geometry (e.g., [6]), but for our purposes we need to develop a slightly different (although equivalent) formulation of the equations, as our surface is defined implicitly.

Let us consider the problem of finding geodesics on the surface defined by

$$f(\mathbf{X},t)=0, \qquad g(\mathbf{X},t)=0. \tag{1}$$

Arbitrary perturbations on the surface satisfy

$$\nabla f\cdot d\mathbf{X}+f_t\,dt=0, \qquad \nabla g\cdot d\mathbf{X}+g_t\,dt=0 \tag{2}$$

where $d\mathbf{X}$ is in the tangential plane to the surface, and thus

$$(g_t\nabla f-f_t\nabla g)\cdot d\mathbf{X}=0. \tag{3}$$

Since $d\mathbf{X}$ is arbitrary in the tangential plane, it follows that $g_t\nabla f-f_t\nabla g$ points normal to the surface.

Now suppose we have a curve $\mathbf{X}(s)$ on the surface, and let $t=t(s)$ be the associated value of t. If

$$\frac{d^2\mathbf{X}}{ds^2}=\lambda(g_t\nabla f-f_t\nabla g), \tag{4}$$

then combining equations (3) and (4) we see that

$$\frac{1}{2}\frac{d}{ds}\left|\frac{d\mathbf{X}}{ds}\right|^2=\frac{d^2\mathbf{X}}{ds^2}\cdot\frac{d\mathbf{X}}{ds}=0.$$

Since the length of $d\mathbf{X}/ds$ is a constant, it follows that s is proportional to arclength and hence that $d^2\mathbf{X}/ds^2$ is the principal normal to the curve $\mathbf{X}(s)$ (see [6, p. 55]). From the definition of a geodesic given above, it now follows that $\mathbf{X}(s)$ is a geodesic on the surface. To determine λ we replace $d\mathbf{X}$ by $d\mathbf{X}/ds$ in (3), differentiate with respect to s, and use (4) to get

$$\lambda=\frac{\sum\limits_{i,j}(g_tf_j-f_tg_j)_i\dfrac{d\mathbf{X}_i}{ds}\dfrac{d\mathbf{X}_j}{ds}+\sum\limits_j(g_tf_j-f_tg_j)_t\dfrac{dt}{ds}\dfrac{d\mathbf{X}_j}{ds}}{-\sum\limits_j(g_tf_j-f_tg_j)^2}. \tag{5}$$

From equations (2) we see that we have two possible equations for dt/ds. It turns out that for our particular choice of $g(\mathbf{X},t)$ we can guarantee that $g_t\neq 0$, and thus we use the equation

$$\frac{dt}{ds}=-\nabla g\cdot\frac{d\mathbf{X}/ds}{g_t}. \tag{6}$$

Substituting this expression into (5), expanding the derivatives and rewriting, then gives

$$\lambda = \frac{\left\{\begin{array}{l} \boldsymbol{\tau}^* \cdot (g_t H_f - f_t H_g + (\nabla g_t)(\nabla f)^T - (\nabla f_t)(\nabla g)^T)\boldsymbol{\tau}^* \\ \quad + (dt/ds)[g_{tt}(\nabla f \cdot \boldsymbol{\tau}^*) - f_{tt}(\nabla g \cdot \boldsymbol{\tau}^*) + g_t(\nabla f_t \cdot \boldsymbol{\tau}^*) - f_t(\nabla g_t \cdot \boldsymbol{\tau}^*)] \end{array}\right\}}{-|g_t \nabla f - f_t \nabla g|^2}$$

where

$$\boldsymbol{\tau}^* = \frac{d\mathbf{X}}{ds}, \quad H_f(i,j) = \frac{\partial^2 f}{\partial X_i \partial X_j}, \quad H_g(i,j) = \frac{\partial^2 g}{\partial X_i \partial X_j}.$$

Using (6), we note that $(-\boldsymbol{\tau}^* \cdot \nabla f_t)(\nabla g \cdot \boldsymbol{\tau}^*) = (\nabla f_t \cdot \boldsymbol{\tau}^*) g_t (dt/ds)$; substituting this into the formula for λ we obtain the following first-order system for a geodesic on the surface defined by equations (1).

$$\frac{d\mathbf{X}}{ds} = \boldsymbol{\tau}^*, \qquad \frac{dt}{ds} = \frac{-\nabla g \cdot \boldsymbol{\tau}^*}{g_t},$$

$$\frac{d\boldsymbol{\tau}^*}{ds} = \frac{\left\{\boldsymbol{\tau}^* \cdot \mathbf{Q}\boldsymbol{\tau}^* + L\dfrac{dt}{ds}\right\}}{-|g_t \nabla f - f_t \nabla g|^2}(g_t \nabla f - f_t \nabla g)$$

where

$$\mathbf{Q} = g_t H_f - f_t H_g + \nabla g_t (\nabla f)^T,$$

and

$$L = g_{tt}(\nabla f \cdot \boldsymbol{\tau}^*) - f_{tt}(\nabla g \cdot \boldsymbol{\tau}^*) + 2g_t(\nabla f_t \cdot \boldsymbol{\tau}^*) - f_t(\nabla g_t \cdot \boldsymbol{\tau}^*).$$

Note that we have not eliminated the auxiliary variable t. Instead we treat it as a dependent variable on the same footing with $\mathbf{X}$.

In the above analysis we used the constraint that the curve remain on the surface to eliminate λ in (4). Thus our method is equivalent to the formulation of the problem as

$$\frac{d^2\mathbf{X}}{ds^2} = \lambda(s)\mathbf{n}(\mathbf{X}(s), t(s))$$

with the constraints

$$f(\mathbf{X}(s), t(s)) = 0, \qquad g(\mathbf{X}(s), t(s)) = 0$$

where $\mathbf{n} = g_t \nabla f - f_t \nabla g$. Direct solution of this implicit system is an alternative approach that would merit further investigation.

3. The special case of a tubular surface. Let $\mathscr{C}$ given by $\mathbf{X}_0(t)$ be a space curve parameterised by arclength, and let $R(t)$ be a corresponding radius function. If

$$f(\mathbf{X}, t) = \frac{1}{2}\{|\mathbf{X} - \mathbf{X}_0(t)|^2 - R(t)^2\},$$

$$g(\mathbf{X}, t) = (\mathbf{X} - \mathbf{X}_0(t)) \cdot \frac{d\mathbf{X}_0(t)}{dt},$$

then the surface $f(\mathbf{X}, t)=0$, $g(\mathbf{X}, t)=0$ will be a tube centred on the curve $\mathscr{C}$ with cross-sectional radius $R(t)$ for every t. Suppose that $\mathscr{C}$ and $R(t)$ are such that the tube does not intersect itself, and suppose further that the radius of curvature of $\mathscr{C}$ is greater than $R(t)$ for every t, i.e., there are no folds in the tube. Then we have

$$\left|(\mathbf{X}-\mathbf{X}_0)\cdot\frac{d\boldsymbol{\tau}}{dt}\right| = R\left|\frac{d\boldsymbol{\tau}}{dt}\right| < 1 \quad \text{where } \boldsymbol{\tau}=\frac{d\mathbf{X}_0(t)}{dt},$$

and thus

$$\begin{aligned} g_t &= (\mathbf{X}-\mathbf{X}_0)\cdot\frac{d\boldsymbol{\tau}}{dt}-\boldsymbol{\tau}\cdot\boldsymbol{\tau} \\ &= (\mathbf{X}-\mathbf{X}_0)\cdot\frac{d\boldsymbol{\tau}}{dt}-1 \quad \text{since } t \text{ is arclength along } \mathscr{C} \\ &\neq 0 \qquad \text{(cf. (6)).} \end{aligned}$$

Substitution into the geodesic equations then gives the following first-order system for geodesics on the tube defined by $\mathbf{X}_0(t)$ and $R(t)$:

$$\text{(7)} \qquad \begin{aligned} &\frac{d\mathbf{X}}{ds}=\boldsymbol{\tau}^*, \qquad \frac{dt}{ds}=\frac{\mu}{1-q}, \\ &\frac{d\boldsymbol{\tau}^*}{ds}=\left\{\frac{\alpha\mu^2+2\mu\nu R\dot{R}-(q-1)^2}{(q-1)^2R^2+R^2\dot{R}^2}\right\}\left(\frac{R\dot{R}}{q-1}\boldsymbol{\tau}+(\mathbf{X}-\mathbf{X}_0)\right) \end{aligned}$$

where

$$q=(\mathbf{X}-\mathbf{X}_0)\cdot\frac{d\boldsymbol{\tau}}{dt}, \qquad r=(\mathbf{X}-\mathbf{X}_0)\cdot\frac{d^2\boldsymbol{\tau}}{dt^2},$$

$$\mu=\boldsymbol{\tau}\cdot\boldsymbol{\tau}^*, \qquad \nu=\frac{d\boldsymbol{\tau}}{dt}\cdot\boldsymbol{\tau}^*,$$

$$\alpha=\dot{R}^2+R\ddot{R}-rR\dot{R}/(q-1)-q+1.$$

4. Initial conditions. In order to solve system (7) we need an initial vector $\mathbf{X}_0^0$, an initial time t_0, and an initial starting direction $\boldsymbol{\tau}_0^*$ (which must be perpendicular to the normal to the surface at t_0). Clearly t_0 can be chosen arbitrarily, and then $\mathbf{X}_0^0$ can be found by calculating a vector $\mathbf{V}$ perpendicular to $\boldsymbol{\tau}(t_0)$ and setting

$$\mathbf{X}_0^0=\mathbf{X}_0(t_0)+R(t_0)\mathbf{V}.$$

We can then calculate $\boldsymbol{\tau}_0^*$ by solving

$$\text{(8)} \qquad \boldsymbol{\tau}_0^*\cdot\nabla f(\mathbf{X}_0(t_0), t_0)=0, \quad \boldsymbol{\tau}_0^*\cdot\boldsymbol{\tau}(t_0)=\xi, \quad |\boldsymbol{\tau}_0^*|^2=1$$

where ξ is a given parameter, controlling how tight the initial wind of the geodesic is. Note that ξ cannot be given arbitrarily. For, let

$$\frac{\boldsymbol{\tau}(t_0)\cdot\nabla f(\mathbf{X}_0(t_0), t_0)}{|\nabla f(\mathbf{X}_0(t_0), t_0)|}=\cos\theta, \qquad \xi=\boldsymbol{\tau}_0^*\cdot\boldsymbol{\tau}(t_0)=\cos\phi.$$

Then, since ϕ has a minimum value when $\boldsymbol{\tau}_0^*$, $\boldsymbol{\tau}(t_0)$ and $\nabla f(\mathbf{X}_0(t_0), t_0)$ are coplanar, and since in this case we must have $\cos\phi=\cos(\theta-(\pi/2))=\sin\theta$, we see that

$$|\xi|\leqq\sqrt{1-\left(\frac{\boldsymbol{\tau}(t_0)\cdot\nabla f(\mathbf{X}_0(t_0),t_0)}{|\nabla f(\mathbf{X}_0(t_0),t_0)|}\right)^2}.$$

5. Determination of $\mathscr{C}$ and $R(t)$. Given a set of data points (θ_i,ϕ_i,t_i), $i=1,n$, we can construct a curve $\mathbf{X}_0(t)$ parameterised by arclength as follows. First, we do a cubic spline fit on θ_i and ϕ_i as functions of t to give us $\theta(t)$ and $\phi(t)$. If we now define $\mathbf{X}_0(t)$ by

$$\frac{d\mathbf{X}_0(t)}{dt}=(\cos\theta\cos\phi,\cos\theta\sin\phi,\sin\theta)$$

we see that $|d\mathbf{X}_0(t)/dt|^2=1$ and thus t is guaranteed to be arclength along the curve. Note that (θ,ϕ) are spherical polar coordinates of the unit tangent vector to $\mathbf{X}_0(t)$. $\mathbf{X}_0(t)$ itself is obtained by integration:

$$\begin{aligned}\mathbf{X}_0(t)&=\mathbf{X}_0(0)+\int_0^t\frac{d\mathbf{X}_0(t')}{dt'}\,dt'\\&=\mathbf{X}_0(0)+\int_0^t(\cos\theta(t')\cos\phi(t'),\cos\theta(t')\sin\phi(t'),\sin\theta(t'))\,dt'\end{aligned}$$

where $\mathbf{X}_0(0)$ can be specified arbitrarily.

However, instead of using θ and ϕ as data points, it is often easier to specify points in space through which the curve should pass (approximately). Given such a set of data points (x_i,y_i,z_i) and corresponding radii r_i, $i=1,n$, we estimate the arclength by straight lines and then do a cubic spline fit on each of x_i, y_i, z_i, and r_i as functions of the estimated arclength. This will give a space curve $\tilde{\mathbf{X}}_0(t)$ and a corresponding radius function $R(t)$. Note however, that t will not be arclength along the curve.

We now use $\tilde{\mathbf{X}}_0(t)$ to generate a new set of data points (θ_i,ϕ_i), $i=1,m$, by solving the equations

$$\frac{d\tilde{\mathbf{X}}_0(t)/dt}{|d\tilde{\mathbf{X}}_0(t)/dt|}=(\cos\theta\cos\phi,\cos\theta\sin\phi,\sin\theta)$$

for θ and ϕ at m specified points along the curve $\tilde{\mathbf{X}}_0(t)$. A cubic spline fit is now done on θ_i and ϕ_i as functions of t to give us $\theta(t)$ and $\phi(t)$, thus reducing this problem to the previous case. Although $\mathscr{C}=\mathbf{X}_0(t)$ does not now pass through the original data points (x_i,y_i,z_i), it will pass close to them and therefore we can effectively control the shape of the final tube.

6. Results. We computed geodesics on two different tubes: tube A, with centreline $\mathscr{C}_A$ and radius function $R_A(t)$ generated by the data

$$\begin{aligned}&n_i=\frac{i-1}{19}\pi\\&x_i=5\cos(n_i)\\&y_i=5\sin(n_i)\qquad\text{for } i=1,20,\\&z_i=3n_i\\&r_i=1+\sin(n_i)\end{aligned}$$

and tube B, with $\mathscr{C}_B$ and $R_B(t)$ generated by the arbitrarily chosen points

$$(x_1, y_1, z_1, r_1) = (5.0, 0.0, 0.0, 1.0),$$
$$(x_2, y_2, z_2, r_2) = (4.0, 1.0, -0.5, 1.0),$$
$$(x_3, y_3, z_3, r_3) = (3.0, 2.0, -1.0, 0.8),$$
$$(x_4, y_4, z_4, r_4) = (2.0, 3.0, -0.5, 1.3),$$
$$(x_5, y_5, z_5, r_5) = (0.0, 4.0, 0.0, 1.7),$$
$$(x_6, y_6, z_6, r_6) = (-1.0, 3.0, 1.0, 2.0),$$
$$(x_7, y_7, z_7, r_7) = (-3.0, 1.5, 1.7, 1.9),$$
$$(x_8, y_8, z_8, r_8) = (-4.0, 0.0, 2.1, 1.7),$$
$$(x_9, y_9, z_9, r_9) = (-5.0, -1.0, 2.3, 1.5).$$

The associated tubes can be plotted by solving

$$f(\mathbf{X}, t) = \frac{1}{2}\{|\mathbf{X} - \mathbf{X}_0(t)|^2 - R(t)^2\} = 0,$$

$$g(\mathbf{X}, t) = (\mathbf{X} - \mathbf{X}_0(t)) \cdot \frac{d\mathbf{X}_0(t)}{dt} = 0$$

for fixed values of t and then plotting the resultant rings (Figs. 1 and 2). Equations (7) were solved using a second-order Runge-Kutta method, and particular geodesics are shown in Figs. 3-7. (The choice of second-order Runge-Kutta is arbitrary; the reader may substitute his favourite method or package.) Figures 4 and 5 show how the geodesic can become trapped in a region around the bulge as we mentioned before; how large a region depends on the initial angle of attack.

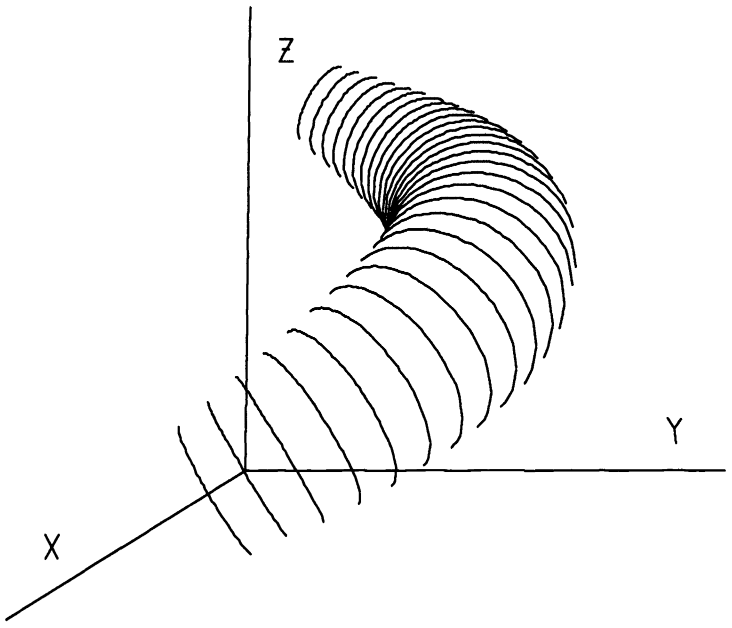


FIG. 1. *Perspective view of tube* A.

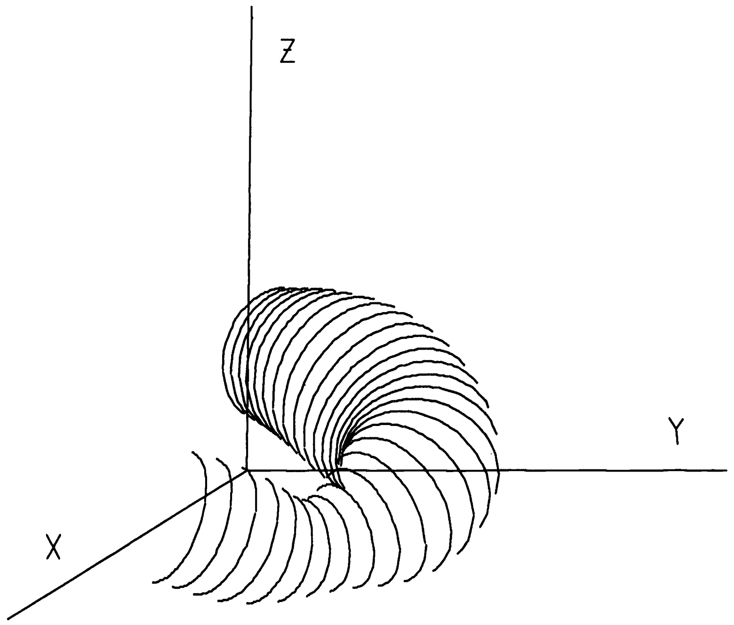


FIG. 2. *Perspective view of tube* B.

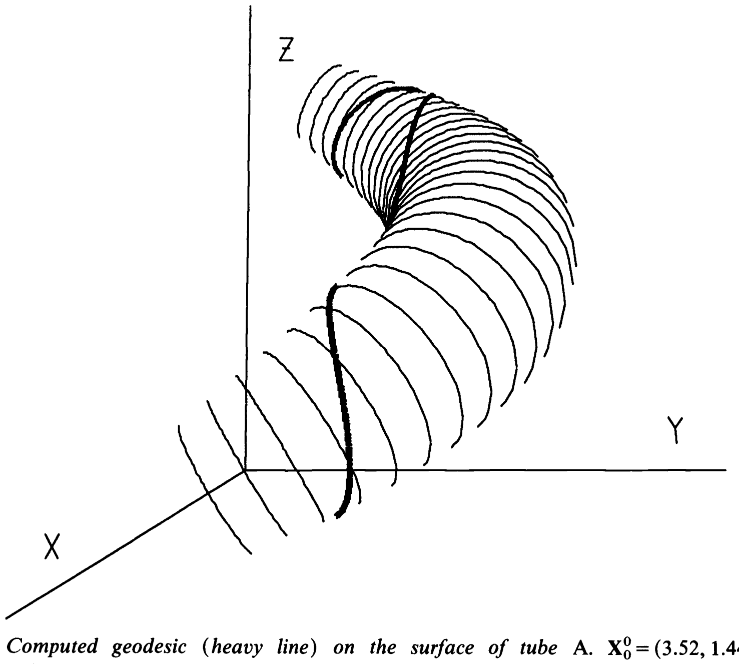


FIG. 3. *Computed geodesic (heavy line) on the surface of tube* A. $X_0^0 = (3.52, 1.44, 0.51)$, $\tau_0^* = (0.18, 0.64, -0.74)$, $\xi = 0.1$, $t_0 = 0.0$. (*See text for definition of these parameters.*)

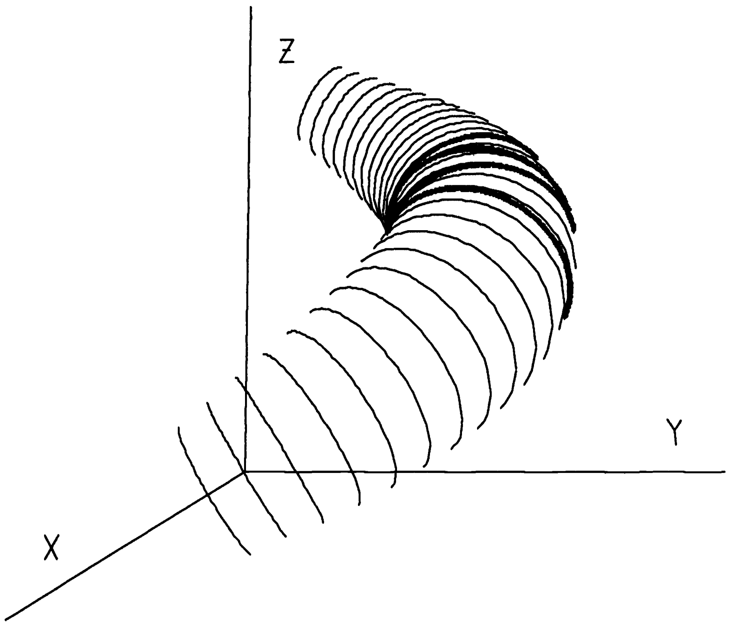


FIG. 4. *Computed geodesic on the surface of tube* A. $\mathbf{X}_0^0 = (-0.95, 3.83, 3.39)$, $\boldsymbol{\tau}_0^* = (0.14, -0.80, 0.58)$, $\xi = 0.2$, $t_0 = 7.5$.

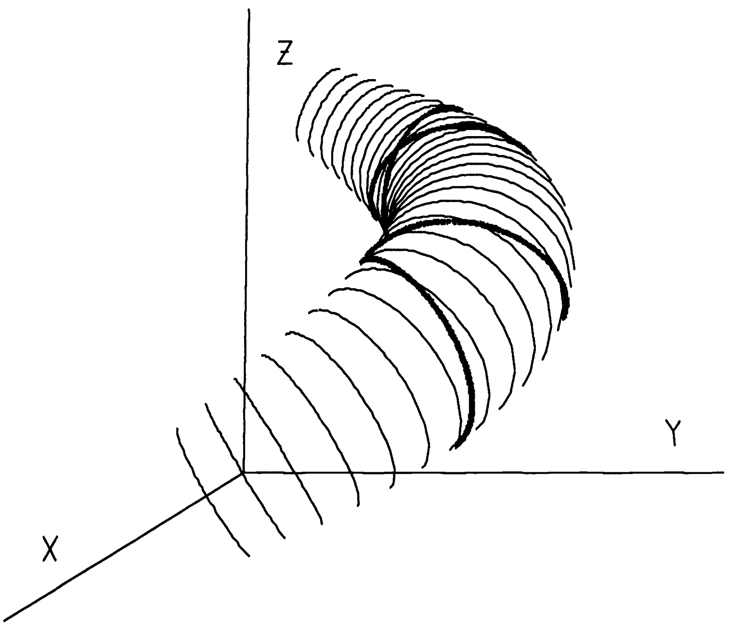


FIG. 5. *Computed geodesic on the surface of tube* A. $\mathbf{X}_0^0 = (-0.95, 3.83, 3.39)$, $\boldsymbol{\tau}_0^* = (-0.15, -0.71, 0.68)$, $\xi = 0.5$, $t_0 = 7.5$.

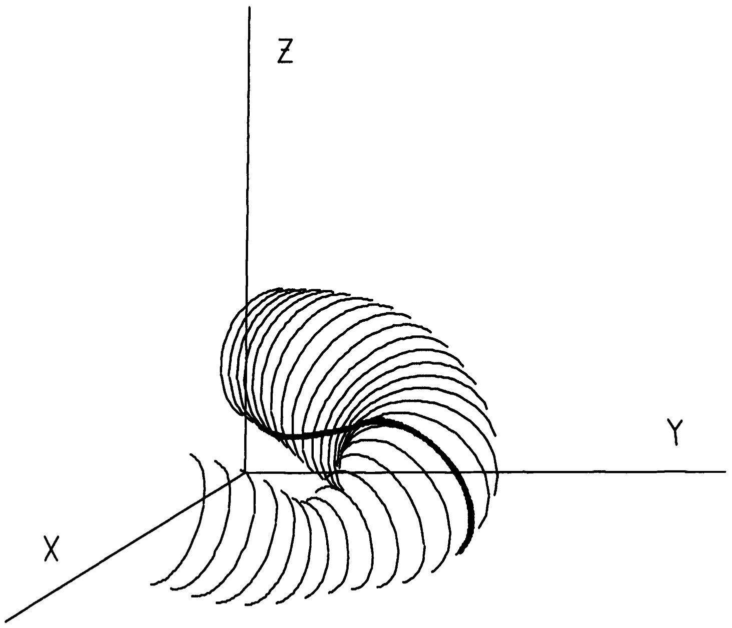


FIG. 6. *Computed geodesic on the surface of tube* B. $\mathbf{X}_0^0 = (-0.31, 2.69, -1.05)$, $\tau_0^* = (0.50, 0.68, -0.54)$, $\xi = -0.7$, $t_0 = 5.0$.

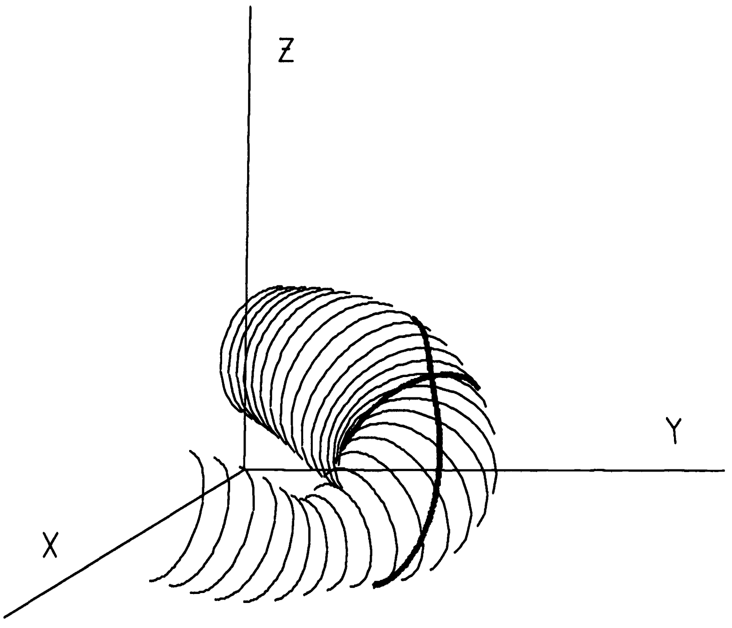


FIG. 7. *Computed geodesic on the surface of tube* B. $\mathbf{X}_0^0 = (-2.60, 1.69, -0.31)$, $\tau_0^* = (-0.25, 0.95, -0.20)$, $\xi = -0.6$, $t_0 = 8.0$.

7. Convergence properties of the numerical method. From our method of generating $\mathscr{C}$ and $R(t)$ from a set of data points, we see that $R(t)$ and $d\mathbf{X}_0(t)/dt$ are piecewise cubic, and hence that the functions r and α of equation (7) are not analytic, having a derivative with discontinuities at each knot of the spline. Therefore we would not necessarily expect a second-order method to exhibit second-order convergence, as the discontinuities of the derivative could introduce extra numerical error. Although we have done no theoretical analysis of the convergence properties of our method, computations suggest that convergence is approximately second order for both the tubes under consideration (see Appendix A). Part of the explanation for this is that discontinuities of the derivative only affect the numerical error at a finite number of points, whereas between knots the convergence will be second order.

8. More general tubes. So far we have only computed geodesics on tubular surfaces with a circular cross section. However, by appropriate choice of f and g our theory is easily extended to deal with more general cases: in particular,

$$f(\mathbf{X}, t) = (\mathbf{X} - \mathbf{X}_0(t))^T[\eta \mathbf{B}(t)\mathbf{B}(t)^T + \mathbf{I}](\mathbf{X} - \mathbf{X}_0(t)) - R(t)^2 = 0,$$
$$g(\mathbf{X}, t) = (\mathbf{X} - \mathbf{X}_0(t)) \cdot \frac{d\mathbf{X}_0(t)}{dt} = 0$$

where $\eta = 1/\gamma^2 - 1$ for γ constant, $\mathbf{B}$ is the binormal to $\mathscr{C}$, and $\mathbf{I}$ is the identity matrix, would give us a tube with elliptical cross section perpendicular to $\boldsymbol{\tau}$, with axes of length R and γR. When $\gamma = 1$, this of course just reduces to the previous case.

More generally, at each point $\mathbf{X}_0(t)$ of $\mathscr{C}$ we could specify an orthogonal set of vectors $\mathbf{U}(t), \mathbf{V}(t)$, and $\mathbf{W}(t)$, and then compute geodesics using

$$f(\mathbf{X}, t) = (\mathbf{X} - \mathbf{X}_0(t))^T[\eta \mathbf{W}(t)\mathbf{W}(t)^T + \mathbf{I}](\mathbf{X} - \mathbf{X}_0(t)) - R(t)^2 = 0,$$
$$g(\mathbf{X}, t) = (\mathbf{X} - \mathbf{X}_0(t)) \cdot \mathbf{U} = 0,$$

which would give us a tube with elliptical cross section perpendicular to $\mathbf{U}$.

9. Conclusions and future work. We now have a method for computing geodesics on an arbitrary non-selfintersecting tube with no folds and a circular cross section perpendicular to the centre curve. The radius of the tube can vary with position, there being no restriction on how rapidly it may vary, and the method is easily extended to tubes with elliptical cross sections perpendicular to a given vector not necessarily the tangent to the centre line.

Our work was motivated by two applications in particular: first, the need to construct the great vessels of the heart out of geodesic fibres and second, the desire to construct a foetal heart at an early stage of development in order to study blood flow in developing hearts. The first application is part of an ongoing effort to model an entire human heart and associated vessels in three dimensions using elastic fibres that follow geodesic or asymptotically geodesic paths [4], [5].

The arrangement of muscle fibres in the adult heart wall has been described by Thomas [7] and by Streeter et al. [8]. A notable feature of the fibre architecture as described by these investigators is a nested family of surfaces on which the fibres follow geodesic curves. A mechanical explanation of this observation appears in the work of Peskin [4]. If we consider the developmental process by which this elaborate fibre architecture is created as the heart changes shape from a simple tube to a thick-walled structure, we may imagine that the phenomenon of geodesic confinement illustrated in this paper plays an important role. This would happen as a tube that is initially cylindrical and reinforced by helical fibres begins to develop a bulge. In the region of the bulge, fibres must (in order to remain geodesics) break off and form a cuff of fibres that are confined to that region. The excess of fibres in the bulge would then reinforce the wall there, the confined fibres constituting the bulk of the ventricular wall.

Unlike the adult heart, the early foetal heart does not consist of muscle fibres arranged in set patterns. However, the outer layer of the early foetal heart, the myocardium, contains large numbers of contractile fibres called myofibrils, and although not much is known about their orientation there are a number of reasons to suppose that they are locally geodesics and can therefore be reasonably modelled by our method. The early foetal heart is a hydrostatically supported tube with internal pressure from both the blood and the cardiac jelly [1], [3] and thus any contractile elements in the myocardium will tend to orient themselves so as to minimise stress (and strain), i.e., along geodesic paths. Such geodesic paths (for instance, the paths given in Figs. 4 and 5) then agree with the experimental evidence that suggests that many of the myofibrils are oriented in a circumferential manner [1], [3] and that the structure of the myofibrils is responsible for the formation of the heart loop. Manasek et al. [2] have postulated that loop formation is governed by the response of anisotropic helical arrangements of myofibrils to internal pressure forces, and such arrangements, reinforcing the inner bend of the tube, are exactly what we have computed (Figs. 3, 6, and 7). We may speculate that this process is self-reinforcing: as the initial anisotropic arrangement of the myofibrils causes the heart to bend in response to internal pressure, the myofibrils will tend to form geodesics on surfaces inside the heart wall, which could then cause further bending.

Appendix A. All the following convergence results were evaluated at $s = 10.0000$, using the initial data:

Tube A	Tube B
$t_0 = 0$	$t_0 = 0$
$\mathbf{X}_0^0 = (3.5229, 1.4376, 0.5144)$	$\mathbf{X}_0^0 = (3.5644, 0.2483, -0.9952)$
$\boldsymbol{\tau}_0^* = (-0.0202, 0.8898, -0.4557)$	$\boldsymbol{\tau}_0^* = (-0.8602, 0.4104, 0.3025)$
$\xi = 0.5$	$\xi = 0.7$

The rate of convergence of $(x(10), y(10), z(10))$ and $t(10)$ is shown in Table 1, where Δx_i means $x(s = 10, \Delta s = 0.1/2^i) - x(s = 10, \Delta s = 0.1/2^{i-1})$. The ratios in Table 1 are close to four, which indicates that the method is approximately second order.

TABLE 1

	Tube A	Tube B		Tube A	Tube B
$\Delta x_2/\Delta x_1 =$	3.9824	4.1461	$\Delta z_2/\Delta z_1 =$	2.1021	3.9351
$\Delta x_3/\Delta x_2 =$	3.9908	3.9896	$\Delta z_3/\Delta z_2 =$	3.2001	3.9063
$\Delta x_4/\Delta x_3 =$	4.0034	3.7419	$\Delta z_4/\Delta z_3 =$	3.6985	3.9891
$\Delta y_2/\Delta y_1 =$	3.4793	4.1874	$\Delta t_2/\Delta t_1 =$	4.1237	4.0379
$\Delta y_3/\Delta y_2 =$	3.7073	4.0847	$\Delta t_3/\Delta t_2 =$	4.0643	3.9363
$\Delta y_4/\Delta y_3 =$	3.9193	3.8130	$\Delta t_4/\Delta t_3 =$	4.0366	3.8534

Tables 2 and 3 show the convergence properties of f and g, again evaluated at $s = 10$. We see from Table 3 that f and g are converging to zero at a rate very similar to the convergence rate of the numerical method. Thus we may conclude that our trajectory is indeed converging to a point on the surface.

TABLE 2

Tube A		
Δs	f	g
0.1	-0.2565×10^{-2}	-0.4144×10^{-3}
0.05	-0.7103×10^{-3}	-0.1041×10^{-3}
0.025	-0.1871×10^{-3}	-0.2626×10^{-4}
0.0125	-0.4817×10^{-4}	-0.6758×10^{-5}
0.00625	-0.1268×10^{-4}	-0.1881×10^{-5}

Tube B		
Δs	f	g
0.1	-0.4482×10^{-2}	0.9250×10^{-3}
0.05	-0.1040×10^{-2}	0.2358×10^{-3}
0.025	-0.2613×10^{-3}	0.5994×10^{-4}
0.0125	-0.7322×10^{-4}	0.1518×10^{-4}
0.00625	-0.1576×10^{-4}	0.3937×10^{-5}

TABLE 3

	Tube A	Tube B		Tube A	Tube B
$f_2/f_1=$	3.6118	4.3180	$g_2/g_1=$	3.9780	3.9224
$f_3/f_2=$	3.7962	3.9791	$g_3/g_2=$	3.9674	3.9342
$f_4/f_3=$	3.8845	3.5685	$g_4/g_3=$	3.8855	3.9486
$\Delta f_2/\Delta f_1=$	3.5459	4.4317	$\Delta g_2/\Delta g_1=$	3.9815	3.9184
$\Delta f_3/\Delta f_2=$	3.7655	4.1390	$\Delta g_3/\Delta g_2=$	3.9958	3.9293
$\Delta f_4/\Delta f_3=$	3.9158	3.2732	$\Delta g_4/\Delta g_3=$	3.9985	3.9811

REFERENCES

[1] F. J. MANASEK, Y. ISOBE, Y. SHIMADA, AND W. HOPKINS, *The embryonic myocardial cytoskeleton, interstitial pressure, and the control of morphogenesis*, in Proc. 2nd Sankei International Symposium, Nora and A. Takao, eds., Futura, New York, 1984, pp. 359-376.

[2] F. J. MANASEK, R. R. KULIKOWSKI, A. NAKAMURA, Q. NGUYENPHUC, AND J. W. LACKTIS, *Early heart development: A new model of cardiac morphogenesis*, in Growth of the Heart in Health and Disease, R. Zak, ed., Raven Press, New York, 1984, pp. 105-130.

[3] A. NAKAMURA, R. R. KULIKOWSKI, J. W. LACKTIS, AND F. J. MANASEK, *Heart looping: A regulated response to deforming forces*, in Etiology and Morphogenesis of Congenital Heart Disease, R. van Praagh and A. Takao, eds., Futura, New York, 1980, pp. 81-98.

[4] C. S. PESKIN, *Fiber-architecture of the left ventricular wall: An asymptotic analysis* CPAM, 42 (1989), pp. 79-113.

[5] C. S. PESKIN AND D. M. MCQUEEN, *A three-dimensional computational method for blood flow in the heart:* (I) *immersed elastic fibers in a viscous incompressible fluid*, J. Comput. Phys., 81 (1989), pp. 372-405.

[6] J. J. STOKER, *Differential Geometry*, John Wiley, New York, 1969.

[7] C. E. THOMAS, *The muscular architecture of the ventricles of hog and dog hearts*, Amer. J. Anat., 101 (1957), pp. 17-57.

[8] D. D. STREETER, JR., W. E. POWERS, M. A. ROSS, AND F. TORRENT-GUASP, *Three-dimensional fiber orientation in the mammalian left ventricular wall*, in Cardiovascular System Dynamics, J. Baan, A. Noordergraaf, and J. Raines eds., M.I.T. Press, Cambridge MA, 1978, pp. 73-84.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 2, pp. 242-262, March 1990

COMPUTATION OF EXPONENTIAL SPLINES*

BRIAN J. McCARTIN†

Abstract. Pruess's investigations [*J. Approx. Theory*, 17 (1976), pp. 86-96], [*Math. Comp.*, 33 (1979), pp. 1273-1281] revealed the shape preservation properties of exponential splines and provided the impetus for further theoretical study of exponential splines [*J. Approx. Theory*, to appear]. Together, these theoretical results form the backdrop for the detailed analysis of issues in the computation of exponential splines contained herewith. Specifically, first and foremost the construction of tension parameter selection algorithms is considered. The conditioning and iterative solution of the spline equations, as well as the derivation and accuracy of end conditions, are discussed. This inquiry concludes with a potpourri of numerical considerations and the presentation of a variety of numerical examples.

Key words. exponential splines, tension splines, shape preserving interpolation, comonotone interpolation, coconvex interpolation

AMS(MOS) subject classifications. 65M05, 65M10

1. Introduction. There is a widespread need for smooth interpolants in the applications [8]. This need was generally satisfied by polynomial interpolation prior to the introduction of polynomial spline functions by Schoenberg [13] in the 1940s. His idea can be condensed as follows. Given $N+1$ data points, locally, i.e., between two consecutive data points, the interpolant is to be a polynomial of degree n while the global interpolant is only required to belong to C^{n-1}. This results in $N(n+1)-(n-1)$ conditions for the determination of $N(n+1)$ polynomial coefficients. That is to say, we have an $(n-1)$-dimensional space of polynomial spline interpolants to the data. Additional constraints are typically provided in the form of end conditions. Note that for $n=1$ we obtain the classical linear splines.

A popular choice is $n=3$, i.e., cubic spline interpolation. The use of such low degree polynomials reduces the risk of wiggles, while second derivative continuity is sufficient in many applications. An additional attraction of the cubic spline is that it possesses a direct analogue in beam theory. This is the draftsman's spline, whence comes the name of this mode of approximation.

Since their introduction, splines have been studied intensively. Convergence of interpolatory splines has been established, as well as convergence of high derivatives provided the function being approximated is sufficiently smooth. In general, the rate of convergence depends on the degree of smoothness of this underlying function.

The practical utility of cubic splines is evidenced by their widespread use as finite-element basis functions, in collocation approximations to differential equations, and in geometric and data-fitting applications. At first glance, it would seem that many issues in practical problems of interpolation and approximation have been resolved by their introduction.

This is true to a limited extent. However, cubic splines can and do produce spurious oscillations in the interpolant. In some cases, this is merely a nuisance but in others it can prove to be detrimental. For example, in combustion calculations it could produce

* Received by the editors December 3, 1984; accepted for publication (in final revised form) February 9, 1989. This work was partially supported by the U.S. Department of Energy, Division of Basic Energy Sciences, Applied Mathematical Sciences Program, under contract DE-AC02-76ERO-3077 and by the National Aeronautics and Space Administration under grants NGT-33-016-800 and NGT-33-016-201.

† Department of Computer and Information Science, The Hartford Graduate Center, Hartford, Connecticut 06120.

an unrealistic detonation, or in computational aerodynamics it could result in the generation of a nonphysical shock wave.

Schweikert [14] first proposed the hyperbolic spline as a remedy to these difficulties. We arrive at the hyperbolic spline by a recourse to beam theory. We add a tensile force that has the effect of pulling the beam taut between the support points. The resulting interpolant includes hyperbolic functions in place of higher-order monomials as basis functions. Späth [16] later considered the general case of a variable tensile force, i.e., the exponential spline. Pruess [10], [11] has rigorously established that for sufficiently great tensile forces the exponential spline so produced mimics both convexity and monotonicity properties present in the data. The theory of exponential splines has been extensively developed and many results are available in the literature [9]-[11].

In the present study, we concern ourselves with matters related to the computation of exponential splines. Our primary goal is to develop new tension parameter selection algorithms that produce co-convex and/or co-monotone interpolants. This complements the nonconstructive existence proofs previously noted. It must be emphasized that the lack heretofore of viable tension parameter selection schemes has greatly diminished the practical utility of exponential splines.

With this crucial issue resolved, we then proceed to an extensive treatment of numerical issues in the computation of exponential splines. We begin this discussion with the derivation of bounds for the condition number of the spline tridiagonal matrices. For the second derivative formulation this was supplied by Pruess [10], whereas for the first derivative formulation this is new (see § 2 for these formulations). There follows an analysis of the iterative solution of the spline equations including a discussion of the optimum relaxation factor. Next is a treatment of spline end conditions. Both the techniques used and the results obtained are unavailable elsewhere.

A variety of numerical topics is then considered. This discussion includes the parametric exponential spline and the periodic exponential spline. An alternative power series representation, as suggested by Pruess [10], is presented that avoids the possible loss of significance in the evaluation of the matrix elements for small arguments of the hyperbolic functions. Also, the need for scaling is considered. We conclude with a sequence of examples that clearly demonstrates both the inherent superiority of exponential splines to cubic splines and the efficacy of our tension parameter selection algorithms.

2. Review of theory. In this section we review the theoretical results on exponential splines [9] that will be of service to us in the remaining pages. We begin with some notation.

$$\begin{aligned}
N &= \text{number of spline intervals},\\
a = x_1 < \cdots < x_{N+1} = b &= \text{spline nodes},\\
h_i\ (i=1,\cdots,N) &= \text{length of } i\text{th spline interval},\\
p_i\ (i=1,\cdots,N) &= \text{tension parameter on } i\text{th spline interval},\\
f &= \text{data},\\
s &= \text{cubic spline interpolant},\\
\tau &= \text{exponential spline interpolant},\\
b_1 &= (f_2-f_1)/h_1 - f'(a),\\
b_i\ (i=2,\cdots,N) &= (f_{i+1}-f_i)/h_i - (f_i - f_{i-1})/h_{i-1}\\
b_{N+1} &= f'(b) - (f_{N+1}-f_N)/h_N\\
s_i\ (i=1,\cdots,N) &= \sinh(p_i h_i)\\
c_i\ (i=1,\cdots,N) &= \cosh(p_i h_i),
\end{aligned}$$

$$\begin{aligned} e_i\ (i=1,\cdots,N) &= (1/h_i - p_i/s_i)/p_i^2 \\ d_i\ (i=1,\cdots,N) &= (p_ic_i/s_i - 1)h_i)/p_i^2, \\ m_0 &= f'(a), \\ m_i\ (i=1,\cdots,N) &= (f_{i+1}-f_i)/h_i, \\ m_{N+1} &= f'(b). \end{aligned}$$

We then have the second derivative formulation on $[x_i, x_{i+1}](i=1,\cdots,N)$:

$$\tau(x) = \frac{1}{p_i^2 s_i}\{\tau_i'' \sinh p_i(x_{i+1}-x) + \tau_{i+1}'' \sinh p_i(x-x_i)\} + \left(f_i - \frac{\tau_i''}{p_i^2}\right)\frac{x_{i+1}-x}{h_i} + \left(f_{i+1} - \frac{\tau_{i+1}''}{p_i^2}\right)\frac{x-x_i}{h_i},$$

where $\tau_i''\ (i=1,\cdots,N+1)$ is the solution of the tridiagonal system

$$\begin{aligned} &d_1\tau_1'' + e_1\tau_2'' = b_1, \\ &e_{i-1}\tau_{i-1}'' + (d_{i-1}+d_i)\tau_i'' + e_i\tau_{i+1}'' = b_i \qquad (i=2,\cdots,N), \\ &e_N\tau_N'' + d_N\tau_{N+1}'' = b_{N+1}. \end{aligned}$$

In the above, we have used specified first derivative end conditions.

We also have the first derivative formulation on $[x_i, x_{i+1}]\ (i=1,\cdots,N)$:

$$\begin{aligned} \tau(x) = {}& f_i \cdot \frac{x_{i+1}-x}{h_i} + f_{i+1} \cdot \frac{x-x_i}{h_i} \\ &+ \frac{1}{e_i-d_i} \cdot \frac{f_{i+1}-f_i}{h_i} \cdot \left[\frac{\sinh p_i(x-x_i) - \sinh p_i(x_{i+1}-x)}{p_i^2 s_i} + \frac{x_{i+1}-2x+x_i}{p_i^2 h_i}\right] \\ &+ \tau_i'\left\{\frac{d_i}{e_i^2-d_i^2}\left[\frac{\sinh p_i(x_{i+1}-x)}{p_i^2 s_i} - \frac{x_{i+1}-x}{p_i^2 h_i}\right] - \frac{e_i}{e_i^2-d_i^2}\left[\frac{\sinh p_i(x-x_i)}{p_i^2 s_i} - \frac{x-x_i}{p_i^2 h_i}\right]\right\} \\ &+ \tau_{i+1}'\left\{\frac{e_i}{e_i^2-d_i^2}\left[\frac{\sinh p_i(x_{i+1}-x)}{p_i^2 s_i} - \frac{x_{i+1}-x}{p_i^2 h_i}\right] - \frac{d_i}{e_i^2-d_i^2}\left[\frac{\sinh p_i(x-x_i)}{p_i^2 s_i} - \frac{x-x_i}{p_i^2 h_i}\right]\right\}, \end{aligned}$$

where $\tau_i'\ (i=1,\cdots,N+1)$ is the solution to the tridiagonal system

$$\begin{aligned} &\left[\frac{d_1}{d_1^2-e_1^2}\right]\tau_1' + \left[\frac{e_1}{d_1^2-e_1^2}\right]\tau_2' = \left[\frac{1}{d_1-e_1}\right] \cdot \left[\frac{f_2-f_1}{h_1}\right] - \tau_1'', \\ &\left[\frac{e_{i-1}}{d_{i-1}^2-e_{i-1}^2}\right]\tau_{i-1}' + \left[\frac{d_{i-1}}{d_{i-1}^2-e_{i-1}^2} + \frac{d_i}{d_i^2-e_i^2}\right]\tau_i' + \left[\frac{e_i}{d_i^2-e_i^2}\right]\tau_{i+1}' \\ &\qquad = \left[\frac{1}{d_{i-1}-e_{i-1}}\right] \cdot \left[\frac{f_i-f_{i-1}}{h_{i-1}}\right] + \left[\frac{1}{d_i-e_i}\right] \cdot \left[\frac{f_{i+1}-f_i}{h_i}\right], \qquad i=2,\cdots,N, \\ &\left[\frac{e_N}{d_N^2-e_N^2}\right]\tau_N' + \left[\frac{d_N}{d_N^2-e_N^2}\right]\tau_{N+1}' = \tau_{N+1}'' + \left[\frac{1}{d_N-e_N}\right] \cdot \left[\frac{f_{N+1}-f_N}{h_N}\right]. \end{aligned}$$

In the above, we have used specified second derivative end conditions.

3. The parameter selection algorithm for co-convex interpolation. In this section and in § 4 we take up the task of tension parameter selection. This is the key problem that needs to be solved if exponential splines are to be useful in practice. A satisfactory treatment is not available in the literature.

Pruess' results [10], [11] assure us that, for large enough tension parameters, the exponential spline interpolant is free from extraneous inflection points. An inflection point is said to be extraneous on $[x_i, x_{i+1}]$ if $b_ib_{i+1}>0$. The question now arises as to how to choose p_i $(i=1, \cdots, N)$ that are sufficiently but not excessively large. Excessively large estimates will produce an interpolant that is kinky in appearance since τ_i'', $\tau_{i+1}''\to\infty$ as $p_k\to\infty$ $(k=i-1, i, i+1)$. In this section we present a tension parameter selection scheme that answers this question in both a theoretical and practical sense. (Note. Assume that $b_i\neq 0$; $i=1, \cdots, N+1$.)

Assume $\tau_i''\neq 0$ $(i=1,\cdots, N+1)$; we then have that $\tau_i''b_i>0$ $(i=1,\cdots,N+1)$ is a necessary and sufficient condition for no extraneous inflection points [14]. Hence, we will iteratively alter p_i $(i=1,\cdots,N)$ so as to enforce $\tau_i''b_i>0$ $(i=1,\cdots,N+1)$. Before proceeding we need the following easily established facts:

(a) $p_i>0 \Rightarrow d_i>0$;

(b) $ab>0$ iff $|a-b|<\max(|a|,|b|)$.

For purposes in illustration, assume that for some choice $p_i^{(n)}$ $(i=1,\cdots,N)$, we have $\tau_k''b_k<0$ for some k between two and N (a similar analysis will subsequently be given for the end intervals). We then have

$$|e_{k-1}\tau_{k-1}''+e_k\tau_{k+1}''| = |b_k-(d_{k-1}+d_k)\tau_k''|.$$

Now

$$b_k\tau_k''>0 \quad \text{iff } |b_k-(d_{k-1}+d_k)\tau_k''|<\max(|b_k|, (d_{k-1}+d_k)|\tau_k''|)$$

since

$$d_i>0 \quad \text{for } p_i>0\ (i=1,\cdots,N).$$

We therefore define $\tilde{p}_{k-1}, \tilde{p}_k$ so that $\tilde{e}_{k-1}, \tilde{e}_k$ produce

$$|\tilde{e}_{k-1}\tau_{k-1}''+\tilde{e}_k\tau_{k+1}''|<\max(|b_k|,(d_{k-1}+d_k)|\tau_k''|);$$

i.e., after freezing the τ'''s, we vary the p's so that all these inequalities are satisfied.

Letting

$$\tilde{e}_i<\frac{\max(|b_k|,(d_{k-1}+d_k)|\tau_k''|)}{2\max(|\tau_{k-1}''|,|\tau_{k+1}''|)}, \qquad i=k-1, k,$$

we have the desired result.

Define

$$\bar{\lambda}=\frac{\max(|b_k|,(d_{k-1}+d_k)|\tau_k''|)}{2\max(|\tau_{k-1}''|,|\tau_{k+1}''|)}.$$

Now, since

$$e_i=\left[\frac{1}{h_i}-\frac{p_i}{\sinh(p_ih_i)}\right]\bigg/p_i^2,$$

we seek to satisfy the inequality

$$g_i(x)<0 \qquad (i=k-1,k),$$

where

$$g_i(x)=\left[\frac{1}{h_i}-\frac{x}{\sinh(h_ix)}\right]\bigg/x^2-\bar{\lambda} \qquad (i=k-1,k).$$

These considerations lead directly to

$$\frac{1}{h_i}-p_i^2\bar{\lambda}<\frac{p_i}{\sinh(p_ih_i)}.$$

This is certainly the case if

$$p_i=(\bar{\lambda}h_i)^{-1/2}.$$

Consequently, we define

$$\tilde{p}_i = (\bar{\lambda} h_i)^{-1/2} \quad \text{(or perhaps } \tilde{p}_i = \max\,((\bar{\lambda} h_i)^{-1/2}, p_i^{(n)})) \quad i = k-1, k$$

(i.e., we always increase the p's).

However, the new tension parameters ($\tilde{p}$'s) will alter the τ'''s, thus requiring an iterative procedure. The suggested procedure is

$$p^{(n+1)} = p^{(n)} + \omega(\tilde{p} - p^{(n)}).$$

It was previously assumed that $\tau_i'' \neq 0$ $(i = 1, \cdots, N+1)$. It can be shown [14] that if $\tau_i'' = 0$ for any $2 \leqq i \leqq N$ then $\tau_{i-1}''\tau_{i+1}'' \geqq 0$ is necessary and sufficient for the prior analysis to hold. Now we have

$$(d_{i-1} + d_i)\tau_i'' = b_i - e_{i-1}\tau_{i-1}'' - e_i\tau_{i+1}'',$$

and, since we still assume $b_k \neq 0$, for all k, we cannot have $\tau_{i-1}'' = \tau_{i+1}'' = 0$ (i.e., either e_{i-1} or e_i is actually present in the equation). So to make $\tau_i'' \neq 0$ when $\tau_{i-1}''\tau_{i+1}'' < 0$, we perturb e_{i-1} and e_i by incrementing p_{i-1} and p_i by some small amount ε.

It remains to remove the assumption $b_k \neq 0$, for all k. If $b_i = 0$, then the points (x_{i-1}, y_{i-1}), (x_i, y_i), and (x_{i+1}, y_{i+1}) should be joined by a straight line (with a similar statement true in the end intervals). The two remaining portions should then be fitted separately using slope end conditions at x_{i-1}^- and x_{i+1}^+ derived from the slope of the straight line segment. This may be accomplished implicitly by the following alteration of the coefficient matrix A. Set as many of the set $\{A_{i-1,i}, A_{i,i+1}, A_{i+1,i}, A_{i,i-1}\}$ as exist equal to zero and then proceed as usual. This produces the desired τ_k'' $(k = 1, \cdots, i-1, i+1, \cdots, N+1)$. The interval $[x_{i-1}, x_{i+1}]$ is then fitted separately with a linear function. The points (x_{i-1}, y_{i-1}) and (x_{i+1}, y_{i+1}) are now to be treated as the right- and left-hand endpoints, respectively, of two distinct exponential splines.

In summary, we note that the tension parameter selection problem is inherently nonlinear (by virtue of the nonlinear occurrence of p_i in the interpolant). It is then hardly surprising that an iterative procedure should suggest itself.

We now return to the problem of parameter selection for the end intervals. The relevant equations are

$$d_1\tau_1'' + e_1\tau_2'' = b_1, \qquad e_N\tau_N'' + d_N\tau_{N+1}'' = b_{N+1}.$$

$\underline{\tau_1''b_1 < 0}$. $|e_1\tau_2''| = |b_1 - d_1\tau_1''|$. Now $b_1\tau_1'' > 0$ if and only if $|b_1 - d_1\tau_1''| < \max\,(|b_1|, d_1|\tau_1''|)$. Therefore define $\tilde{p}_1 \ni \tilde{e}_1$ produces $|\tilde{e}_1\tau_2''| < \max\,(|b_1|, d_1|\tau_1''|)$. Letting $\tilde{e}_1 < \max\,(|b_1|, d_1|\tau_1''|)/|\tau_2''|$, we obtain the desired result. Hence, define $\bar{\lambda} = \max\,(|b_1|, d_1|\tau_1''|)/|\tau_2''|$.

$\underline{\tau_{N+1}''b_{N+1} < 0}$. $|e_N\tau_N''| = |b_{N+1} - d_N\tau_{N+1}''|$. Now $b_{N+1}\tau_{N+1}'' > 0$ if and only if $|b_{N+1} - d_N\tau_{N+1}''| < \max\,(|b_{N+1}|, d_N|\tau_{N+1}''|)$. Therefore define $\tilde{p}_N \ni \tilde{e}_N$ produces $|\tilde{e}_N\tau_N''| < \max\,(|b_{N+1}|, d_N|\tau_{N+1}''|)$. Letting $\tilde{e}_N < \max\,(|b_{N+1}|, d_N|\tau_{N+1}''|)/|\tau_N''|$, we obtain the desired result. Hence, define $\bar{\lambda} = \max\,(|b_{N+1}|, d_N|\tau_{N+1}''|)/|\tau_N''|$.

These considerations are codified in Algorithm COCONVEX in the Appendix.

4. The parameter selection algorithm for co-monotone interpolation. In this section we develop a tension parameter selection algorithm that preserves any monotonicity present in the data [6]. Specifically, if the polygonal interpolant has slope of constant sign in three successive intervals, then we select the tension parameters in these intervals so that $\tau'(x)$ has this same sign in the middle interval. Similar considerations apply at the end intervals. The existence of such tension parameters is guaranteed by results of Pruess [10], [11].

We begin by rewriting the exponential spline as $\tau(x)=A_i+B_ix+C_i\,e^{p_ix}+D_i\,e^{-p_ix}$ on $[x_i, x_{i+1}]$. Assume that $\tau'_1\tau'_{i+1}\geqq 0$ with both having the "correct" sign. Therefore, any extremum of $\tau'(x)$ interior to $[x_i, x_{i+1}]$ is characterized by $\tau''(x^*)=0\Rightarrow e^{2p_ix^*}=-D_i/C_i$. Now if $D_i/C_i\geqq 0$ there is no interior extremum and this spline segment is monotonicity preserving. If $D_i/C_i<0$ then $e^{p_ix^*}=\sqrt{-D_i/C_i}\Rightarrow\tau'(x^*)=B_i-2\operatorname{sgn}(D_i)p_i\sqrt{-C_iD_i}$. Also, $\tau'''(x^*)=-2\operatorname{sgn}(D_i)p_i^3\sqrt{-C_iD_i}$. Therefore,

(i) $C_i>0$, $D_i<0\Rightarrow\tau'''(x^*)>0\Rightarrow\tau'(x^*)$ is a (local) minimum;

(ii) $C_i<0$, $D_i>0\Rightarrow\tau'''(x^*)<0\Rightarrow\tau'(x^*)$ is a (local) maximum.

If $\tau'(x^*)$ has the correct sign then $\tau(x)$ is locally monotonicity preserving. If $\tau'(x^*)$ is of the "wrong" sign then we must do something about it if $x^*\in[x_i, x_{i+1}]$. Since

$$x^*=x_i+\frac{1}{2p_i}\ln\left[\frac{\tau''_{i+1}-\tau''_i\,e^{p_ih_i}}{\tau''_{i+1}-\tau''_i\,e^{-p_ih_i}}\right]$$

we thus have two cases:

(a) If $\tau''_i<e^{p_ih_i}\tau''_{i+1}$ then $x^*\in[x_i, x_{i+1}]$ if and only if $\tau''_{i+1}\geqq 0$;

(b) If $\tau''_i>e^{p_ih_i}\tau''_{i+1}$ then $x^*\in[x_i, x_{i+1}]$ if and only if $\tau''_{i+1}\leqq 0$.

We now come to the case in which $\tau'(x^*)$ has the wrong sign. In this event we iteratively modify the tension parameter p_i, so as to enforce the requirement that $\tau'(x^*)$ should have the correct sign.

Consequently, consider once again our special cases, with $\delta>0$, and $\gamma\equiv B_i+p_i\cdot 2\operatorname{sgn}(C_i)\cdot\sqrt{-C_iD_i}$:

(i) $C_i>0$, $D_i<0\Rightarrow\tau'(x^*)$ is a minimum

(a) if $m_i\leqq 0$ then do not alter p_i

(b) if $m_i>0$ then, since we want $\gamma>0$, set

$$p_i^{(n+1)}=-B_i/2\sqrt{-C_iD_i}+\delta.$$

(ii) $C_i<0$, $D_i>0\Rightarrow\tau'(x^*)$ is a maximum

(a) if $m_i\geqq 0$ then do not alter p_i

(b) if $m_i<0$ then, since we want $\gamma<0$, set

$$p_i^{(n+1)}=B_i/2\sqrt{-C_iD_i}+\delta.$$

The only remaining issue is if the computed τ'_i should be of the wrong sign, i.e., $\tau'_im_{i-1}<0$ and $\tau'_im_i<0$.

We observe that

$$\tau'_i=\tfrac{1}{2}\{[e_{i-1}\tau''_{i-1}+(d_{i-1}-d_i)\tau''_i-e_i\tau''_{i+1}]+[m_{i-1}+m_i]\}$$

and enforce

$$e_{i-1}\cdot|\tau''_{i-1}|+(d_{i-1}+d_i)\cdot|\tau''_i|+e_i\cdot|\tau''_{i+1}|<|m_{i-1}+m_i|.$$

Now,

$$e_{i-1}\cdot|\tau''_{i-1}|+(d_{i-1}+d_i)\cdot|\tau''_i|+e_i\cdot|\tau''_{i+1}|\leqq(e_{i-1}+d_{i-1}+e_i+d_i)\max(|\tau''_{i-1}|,|\tau''_i|,|\tau''_{i+1}|).$$

We need the following lemma.

LEMMA. (a) $e_i\leqq 1/p_{i_2}h_i$; (b) $d_i\leqq 1/p_i$.

We are thus led to enforcing

$$\left[\frac{1}{p_{i-1}^2h_{i-1}}+\frac{1}{p_{i-1}}+\frac{1}{p_i^2h_i}+\frac{1}{p_i}\right]\max(|\tau''_{i-1}|,|\tau''_i|,|\tau''_{i+1}|)<|m_{i-1}+m_i|$$

or, alternatively

$$\frac{1}{p_{i-1}^2h_{i-1}}+\frac{1}{p_{i-1}}<\frac{|m_{i-1}+m_i|}{2\max(|\tau''_{i-1}|,|\tau''_i|,|\tau''_{i+1}|)}$$

together with

$$\frac{1}{p_i^2 h_i}+\frac{1}{p_i}<\frac{|m_{i-1}+m_i|}{2\max(|\tau''_{i-1}|,|\tau''_i|,|\tau''_{i+1}|)}.$$

This ultimately leads to

$$p_i^{(n+1)}=\frac{1+p_i h_i}{p_i h_i}\cdot\frac{2\max(|\tau''_{i-1}|,|\tau''_i|,|\tau''_{i+1}|)}{|m_{i-1}+m_i|}+\delta$$

and

$$p_{i-1}^{(n+1)}=\frac{1+p_{i-1}h_{i-1}}{p_{i-1}h_{i-1}}\cdot\frac{2\max(|\tau''_{i-1}|,|\tau''_i|,|\tau''_{i+1}|)}{|m_{i-1}+m_i|}+\delta.$$

These considerations are codified in Algorithm COMONOTONE in the Appendix.

5. Direct solution of the spline equations. As we have seen, the formulation of the spline equations in terms of either first or second derivatives leads to a (symmetric) tridiagonal system. These equations share the property with their cubic spline counterparts that they can be solved in $O(N)$ arithmetic operations, e.g., by the Thomas algorithm [2].

A quantity of interest in the present context is the condition number of the spline matrix A [10]. Let us first take the first derivative formulation. Gerschgorin's theorem reveals that

$$\|A\|_2\leqq 2\max_i\frac{p_i^2 s_i h_i}{p_i h_i c_i-2s_i+p_i h_i},\qquad \|A^{-1}\|_2\leqq\frac{1}{2}\max_i\frac{c_i-1}{p_i s_i}.$$

For the second derivative formulation the same line of reasoning yields

$$\|A\|_2\leqq 2\max_i\frac{c_i-1}{p_i s_i},\qquad \|A^{-1}\|_2\leqq\frac{1}{2}\max_i\frac{p_i^2 s_i h_i}{p_i h_i c_i-2s_i+p_i h_i}.$$

Power series expansions establish

$$\frac{c_{i-1}}{p_i s_i}\leqq\frac{h_i}{2},\qquad \frac{p_i^2 h_i s_i}{p_i h_i c_i-2s_i+p_i h_i}\leqq\frac{6}{h_i}.$$

We thus have the following:

(i) First derivative formulation

$$\|A\|_2\leqq\frac{12}{h_{\min}},\qquad \|A^{-1}\|_2\leqq\frac{h_{\max}}{4};$$

(ii) Second derivative formulation

$$\|A\|_2\leqq h_{\max},\qquad \|A^{-1}\|_2\leqq\frac{3}{h_{\min}}.$$

Hence in both cases we have the matrix condition number

$$\kappa(A)\equiv\|A\|_2\|A^{-1}\|_2\leqq\frac{3h_{\max}}{h_{\min}}.$$

Pruess [10] supplied this result for the second derivative formulation while that for the first derivative formulation is new.

6. Iterative solution of the spline equations. The spline equations in terms of second derivatives with slope end conditions are $A\tau''=b$ with the obvious definitions. Let $A=D+O$ where D is diagonal and $O_{ii}=0$, for all i. Then $(D+O)\tau''=b$, which implies that $\tau''=-D^{-1}O\tau''+D^{-1}b$. Define $R\equiv D^{-1}O$ and $r\equiv D^{-1}b$. Therefore, $\tau''=-R\tau''+r$. In any iterative scheme [5] derived from this relation, due consideration must be given to the eigenvalues of R. Note that, although R is not symmetric, we do have

$$D^{1/2}RD^{-1/2}=D^{-1/2}OD^{-1/2},$$

which is symmetric. Hence, R is similar to a symmetric matrix thus implying that its eigenvalues are real. Moreover, $Rx=\lambda x \Rightarrow R\bar{x}=-\lambda\bar{x}$ where $\bar{x}$ is obtained from x by altering the sign of every other element.

Gerschgorin's theorem then allows us to conclude that all of R's eigenvalues satisfy

$$|\lambda|\leqq \max\left\{\frac{e_1}{d_1}, \frac{e_{i-1}+e_i}{d_{i-1}+d_i}, \frac{e_N}{d_N}\right\}\equiv \mu \leqq \frac{1}{2}.$$

Young's theory then allows us to use simultaneous overrelaxation

$$[\tau'']^{(n+1)}=\omega[r-R[\tau'']^{(n)}]-(\omega-1)[\tau'']^{(n)}$$

and the optimum relaxation factor will be given by

$$\omega^*=\frac{2}{1+\sqrt{1-\lambda_{\max}^2}}.$$

As an estimate of $\lambda_{\max}$ we use μ. To see that this is a reasonable choice, consider the case of uniform mesh width and tension. In this case $\mu=e/d$ and $Rx=\mu x$ with $x=[1, \_\_\_\_, 1]^T$. That is, $\mu=\lambda_{\max}$ and this produces an exact value of ω^*. In this instance, the special case of the cubic spline yields $\mu=\frac{1}{2}$ and $\omega^*=4(2-\sqrt{3})$.

A possible starting point for the iterative process would be

$$[\tau'']^{(0)}=\frac{2b_i}{h_{i-1}+h_i}.$$

The spline equations in terms of first derivatives with second derivative end conditions are $A\tau'=b$ with the obvious definitions.

The iterative treatment of this system proceeds along precisely the same lines as before. However, now

$$|\lambda|\leqq \max\left\{\frac{e_1}{d_1}, \frac{e_{i-1}(d_i^2-e_i^2)+e_i(d_{i-1}^2-e_{i-1}^2)}{d_{i-1}(d_i^2-e_i^2)+d_i(d_{i-1}^2-e_{i-1}^2)}, \frac{e_N}{d_N}\right\}\equiv \mu \leqq \frac{1}{2}.$$

Once again, μ is used as our estimate of $\lambda_{\max}$ with the familiar justification. The above iteration requires only a constant amount of storage in addition to that required for the input data.

7. End conditions. In our previous discussions we have always assigned one boundary condition at each end of the interval of interpolation, i.e., a boundary value problem. It occurs to us however that we could rightfully assign both end conditions at one end, i.e., an initial value problem. This would allow us to concatenate sequentially defined splines in a C^2 fashion. To facilitate clarity we will restrict ourselves to uniform mesh and tension.

If we choose to specify $\tau_1' = \phi'$ and $\tau_1'' = \phi''$ then defining

$$a_i = \frac{\partial \tau_i''}{\partial \phi'}, \qquad A_i = \frac{\partial \tau_i''}{\partial \phi''},$$

$$a_1 = 0, \quad a_2 = -\frac{1}{e}, \quad A_1 = 1, \quad A_2 = -\frac{d}{e},$$

we have

$$ea_{i-2} + 2da_{i-1} + ea_i = 0, \qquad eA_{i-2} + 2dA_{i-1} + eA_i = 0.$$

Thus we consider the difference equation

$$z_{k-1} + 2\mu z_k + z_{k+1} = 0, \quad z_0, z_1 \text{ given},$$

with $\mu \equiv d/e$. Assuming a solution of the form $z_k = aq^k \Rightarrow 1 + 2\mu q + q^2 = 0 \Rightarrow q_1 = -\mu - \sqrt{\mu^2 - 1} < -1$, $q_2 = -\mu + \sqrt{\mu^2 - 1} < 1$. We introduce the basis sequences defined by

$$x_{k-1} + 2\mu x_k + x_{k+1} = 0, \quad x_0 = 1, \quad x_1 = 0,$$

$$y_{k-1} + 2\mu y_k + y_{k+1} = 0, \quad y_0 = 0, \quad y_1 = 1,$$

yielding

$$x_n = \frac{q_2}{q_2 - q_1} q_1^n - \frac{q_1}{q_2 - q_1} q_2^n, \qquad y_n = -\frac{1}{q_2 - q_1} q_1^n + \frac{1}{q_2 - q_1} q_2^n.$$

Hence, $z_n = z_0 \cdot x_n + z_1 \cdot y_n$ and therefore $a_n = -(1/e)y_{n-1}$, $A_n = x_{n-1} - (d/e)y_{n-1}$. We now see why such a specification is not suitable. As $|q_1| > 1$, we have the effect of a perturbation in the initial conditions increasing exponentially. In this sense, these end conditions are ill-conditioned.

Let us use this technique to study the specification of second derivatives at the two ends. Letting $a_i \equiv \partial \tau_i'' / \partial \phi''$, $A_i \equiv \partial \tau_i'' / \partial \psi''$, we have

$$a_1 = 1, \quad a_{N+1} = 0, \quad A_1 = 0, \quad A_{N+1} = 1, \quad \text{and}$$

$$a_{i-1} + 2\mu a_i + a_{i+1} = 0, \qquad A_{i-1} + 2\mu A_i + A_{i+1} = 0.$$

Therefore, $a_n = c_1 q_1^{n-1} + c_2 q_2^{n-1}$, $A_n = k_1 q_1^{n-1} + k_2 q_2^{n-1}$ yielding

$$a_n = \frac{q_1^{N-n+1} - q_2^{N-n+1}}{q_1^N - q_2^N}, \qquad A_n = \frac{q_1^{n-1} - q_2^{n-1}}{q_1^N - q_2^N},$$

which are clearly well behaved.

Now consider the specification of first derivatives at both ends.

Letting $a_i \equiv \partial \tau_i'' / \partial \phi'$, $A_i \equiv \partial \tau_i'' / \partial \psi'$, we have

$$\mu a_1 + a_2 = -\frac{1}{e}, \quad a_N + \mu a_{N+1} = 0, \quad \mu A_1 + A_2 = 0, \quad A_N + \mu A_{N+1} = \frac{1}{e}$$

and $a_{i-1} + 2\mu a_i + a_{i+1} = 0$, $A_{i-1} + 2\mu A_i + A_{i+1} = 0$. Once again, $a_n = c_1 q_1^{n-1} + c_2 q_2^{n-1}$, $A_n = k_1 q_1^{n-1} + k_2 q_2^{n-1}$. Where now

$$c_1 = \frac{1}{e} \cdot \frac{q_2^{N-1} + \mu q_2^N}{[(\mu + q_1)(q_2^{N-1} + \mu q_2^N) + (\mu + q_2)(q_1^{N-1} + \mu q_1^N)]},$$

$$c_2 = -\frac{1}{e} \cdot \frac{q_1^{N-1} + \mu q_1^N}{[(\mu + q_1)(q_2^{N-1} + \mu q_2^N) + (\mu + q_2)(q_1^{N-1} + \mu q_1^N)]},$$

$$k_1 = -\frac{1}{e} \cdot \frac{\mu + q_2}{[(\mu + q_1)(q_2^{N-1} + \mu q_2^N) - (\mu + q_2)(q_1^{N-1} + \mu q_2^N)]},$$

$$k_2 = \frac{1}{e} \cdot \frac{\mu + q_1}{[(\mu + q_2)(q_2^{N-1} + \mu q_2^N) - (\mu + q_2)(q_1^{N-1} + \mu q_1^N)]}.$$

These formulae are of great utility in the following application. Suppose that we spline fit a collection of data and then decide that the end conditions that we used were inappropriate. Rather than computing a new spline, we may use the above to update $\{\tau_i''\}_{i=1}^{N+1}$.

For example, consider the case of second derivatives specified at both ends. The old spline equations are

$$A\tau''=b$$

and the new spline equations are

$$A(\tau''+\delta\tau'')=b+\delta b,$$

where $\delta b=[\delta\phi''\,0\cdots 0\ \ \delta\psi'']^T$. So we may solve $A\cdot\delta\tau''=\delta b$ and add the result to τ''. Here $\delta\tau''$ is simply a linear combination of the previously derived quantities. Specifically, $(\delta\tau'')_i=\delta\phi''\cdot a_i+\delta\psi''\cdot A_i$ so that the spline equations need not be re-solved and no additional storage is required.

Similar comments apply to the case of first derivatives specified. Moreover, all the preceding analysis may be extended in a straightforward fashion to the case of mixed end conditions.

It is interesting to note that if we are confronted with a spline code that only accepts either $\tau_1'=\tau_{N+1}'=0$ or $\tau_1''=\tau_{N+1}''=0$ as end conditions, we can use the following auxiliary functions [3]:

$$A_1(x)=f(x)-\frac{1}{2}\left[\frac{(x-a)^2}{b-a}f'(b)-\frac{(b-x)^2}{b-a}f'(a)\right],$$

$$A_2(x)=f(x)-\frac{1}{6}\left[\frac{(x-a)^3}{b-a}f''(b)+\frac{(b-x)^3}{b-a}f''(a)\right].$$

We then calculate A_i at the knots, spline fit these values using the appropriate end conditions, and then set

$$f(x)\approx\tau(x)+\frac{1}{2}\left[\frac{(x-a)^2}{b-a}f'(b)-\frac{(b-x)^2}{b-a}f'(a)\right]$$

or

$$f(x)\approx\tau(x)+\frac{1}{6}\left[\frac{(x-a)^3}{b-a}f''(b)+\frac{(b-x)^3}{b-a}f''(a)\right],$$

depending on which end conditions were used.

We conclude this section with a treatment of the important practical problem of supplying end conditions when only data points themselves are available.

Let $f\in C^4[x_1,x_{N+1}]$ with $M\equiv\sup|f^{(4)}|$. A theorem of Pruess [10] states that

$$\|D^i(s-\tau)\|\leqq\tfrac{26}{3}p_{\max}^2h^{4-i}\max_j|s_j''|,\qquad i=0,1,2.$$

A result of Kershaw [7] states that

$$\|D^i(f-s)\|\leqq c_i\cdot h^{2-i}[h^2M+8\max_j|f_j''-s_j''|],\qquad i=0,1,2.$$

Hence, we may conclude that

$$\|D^i(\tau-f)\|\leqq\tfrac{26}{3}p_{\max}^2h^{4-i}\max_j|s_j''|+c_ih^{4-i}\cdot M+c_ih^{2-i}\cdot 8\max_j|f_j''-s_j''|,\qquad i=0,1,2.$$

Thus, in order to maintain uniform $O(h^{4-i})$ accuracy, we need only ensure that $\max_j |f_j'' - s_j''| = O(h^2)$. This is well known to be the case when exact slope end conditions are specified. We now investigate the effect of using approximate rather than exact slope end conditions thus extending known results [15] for the cubic spline to the exponential spline.

Defining $As'' = 6b$ and $A\bar{s}'' = 6\bar{b}$, where $\bar{b}$ is obtained from b by replacing f_1' and f_{N+1}' by their estimated values $\bar{f}_1'$ and $\bar{f}_{N+1}'$, results in

$$\|s'' - \bar{s}''\| \leqq 6 \max\left\{\frac{|f_1' - \bar{f}_1'|}{h_1}, \frac{|f_{N+1}' - \bar{f}_{N+1}'|}{h_N}\right\}.$$

If we have

$$\bar{f}_1' = f_1' + O(h^3) \quad \text{and} \quad \bar{f}_{N+1}' = f_{N+1}' + O(h^3),$$

then

$$\max_j |f_j'' - \bar{s}_j''| \leqq \max_j |f_j'' - s_j''| + \max_j |s_j'' - \bar{s}_j''| = O(h^2),$$

as desired.

So we see that we need only supply a third-order accurate approximation to $f'(a)$ and $f'(b)$ in order to retain the interior error bounds. This can be achieved by using the slope predicted by the four-point one-sided difference formula derived from Lagrange interpolation.

This gives us the following end conditions:

$$\bar{f}_1' = c_1 f_1 + c_2 f_2 + c_3 f_3 + c_4 f_4 \quad \text{with } c_1 = -(c_2 + c_3 + c_4), \quad \text{where}$$

$$c_2 = \frac{(h_1 + h_2)(h_1 + h_2 + h_3)}{(h_1)(h_2)(h_2 + h_3)}, \qquad c_3 = -\frac{(h_1)(h_1 + h_2 + h_3)}{(h_1 + h_2)(h_2)(h_3)},$$

$$c_4 = \frac{(h_1)(h_1 + h_2)}{(h_1 + h_2 + h_3)(h_2 + h_3)(h_3)}.$$

$$\bar{f}_{N+1}' = -d_4 f_{N-2} - d_3 f_{N-1} - d_2 f_N - d_1 f_{N+1} \quad \text{with } d_1 = -(d_2 + d_3 + d_4), \quad \text{where}$$

$$d_2 = \frac{(h_N + h_{N-1})(h_N + h_{N-1} + h_{N-2})}{(h_N)(h_{N-1})(h_{N-1} + h_{N-2})}, \qquad d_3 = -\frac{(h_N)(h_N + h_{N-1} + h_{N-2})}{(h_N + h_{N-1})(h_{N-1})(h_{N-2})},$$

$$d_4 = \frac{(h_N)(h_N + h_{N-1})}{(h_N + h_{N-1} + h_{N-2})(h_{N-1} + h_{N-2})(h_{N-2})}.$$

We have the following error estimates:

$$f_1' - \bar{f}_1' = -\frac{f^{(4)}(\xi)}{4!}(h_1)(h_1 + h_2)(h_1 + h_2 + h_3), \qquad \xi \in [x_1, x_4],$$

$$f_{N+1}' - \bar{f}_{N+1}' = \frac{f^{(4)}(\xi)}{4!}(h_N)(h_N + h_{N-1})(h_N + h_{N-1} + h_{N-2}), \qquad \xi \in [x_{N-2}, x_{N+1}],$$

which are easily established by application of Peano's theorem.

On a uniform mesh with constant tension, there is an increase in the order of approximation. Hence, in this instance we should provide higher-order estimates of the end conditions. We simply use

$$\hat{f}'_1=\frac{1}{24h}[-50f_1+96f_2-72f_3+32f_4-6f_5],$$

$$\hat{f}'_{N+1}=\frac{1}{24h}[6f_{N-2}-32f_{N-2}+72f_{N-1}-96f_N+50f_{N+1}],$$

which derive from a quartic Lagrange interpolation [1]. The error estimates are

$$f'_1-\hat{f}'_1=\tfrac{1}{5}h^4f^{(5)}(\xi),\qquad \xi\in[x_1,x_5]$$

and

$$f'_{N+1}+\hat{f}'_{N+1}=\tfrac{1}{5}h^4f^{(5)}(\xi),\qquad \xi\in[x_{N-3},x_{N+1}].$$

The above considerations for first derivative end conditions with second derivatives as unknowns are easily extendable to second derivative end conditions. Furthermore, both of these cases may also be applied to the equations with first derivatives as unknowns with similar results.

8. Numerical considerations. In this section, we detail the necessary modifications and extensions to the above analysis that practical implementation mandates [4], [12].

8.1. Alternative power series representation. The first problem to be treated is the loss of significance when evaluating d_i, e_i, and $\tau(x)$ for small values of p_ih_i (where small is machine dependent). We proceed as follows:

$$e_i=\frac{h_i}{6}\left(1-\frac{7}{60}p_i^2h_i^2+\frac{31}{2520}p_i^4h_i^4\right)+O(p_i^6h_i^7),$$

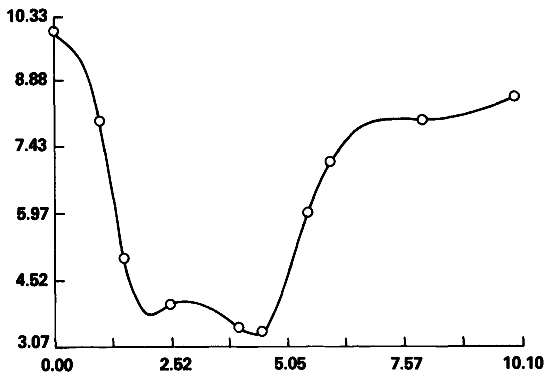
$$d_i=\frac{h_i}{3}\left(1-\frac{1}{15}p_i^2h_i^2+\frac{2}{315}p_i^4h_i^4\right)+O(p_i^6h_i^7).$$

We have the following expansion for $\tau(x)$:

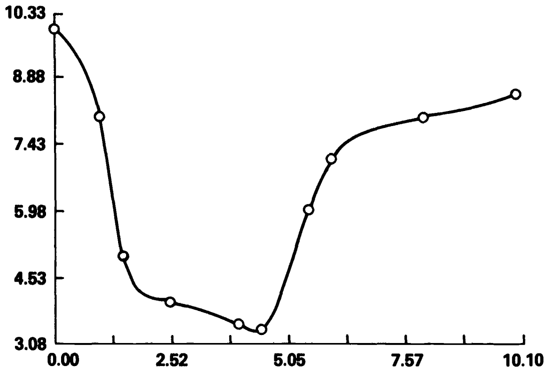
$$\begin{aligned}\tau(x)=f_i\left(\frac{x_{i+1}-x}{h_i}\right)&+f_{i+1}\left(\frac{x-x_i}{h_i}\right)-\frac{h_i}{6}(x_{i+1}-x)\tau''_i\\ &\cdot\left(1-\frac{7}{60}p_i^2h_i^2+\frac{31}{2520}p_i^4h_i^4+\frac{1}{6}p_i^2(x_{i+1}-x)^2-\frac{1}{h_i^2}(x_{i+1}-x)^2\right.\\ &\left.-\frac{7}{195}p_i^4h_i^2(x_{i+1}-x)^2-\frac{p_i^2}{20h_i^2}(x_{i+1}-x)^4+\frac{p_i^4}{120}(x_{i+1}-x)^4\right)\\ &-\frac{h_i}{6}(x-x_i)\tau''_{i+1}\left(1-\frac{7}{60}p_i^2h_i^2+\frac{31}{2520}p_i^4h_i^4+\frac{1}{6}p_i^2(x-x_i)^2-\frac{1}{h_i^2}(x-x_i)^2\right.\\ &\left.-\frac{7}{195}p_i^4h_i^2(x-x_i)^2-\frac{p_i^2}{20h_i^2}(x-x_i)^4+\frac{p_i^4}{120}(x-x_i)^4\right).\end{aligned}$$

A convenient choice for initial tension parameter values is $p_i=0$, for all i, since the above then reduces to the cubic spline.

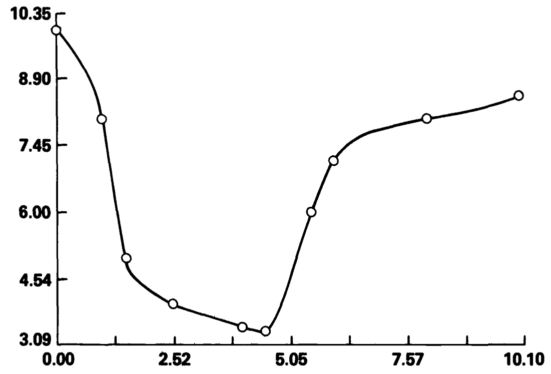
8.2. Scaling. The next problem to be treated is that of the inherent number range restriction. This results in the restriction that the magnitude of exponential arguments be less than σ, which is a machine dependent constant. Since the exponential spline definition requires the computation of quantities such as a sinh (p_ih_i), the data must be scaled so that $\max_{1\leq i\leq N}\{p_ih_i\}<\sigma$. It is easily shown that a scaling of the abscissae leaves the sign of τ''_i $(i=1,\cdots,N+1)$ invariant. This is required since the necessary



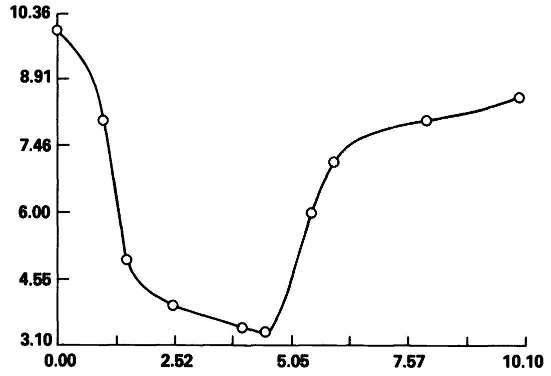
(a) Cubic spline, iteration 0.



(b) Exponential spline, iteration 1.

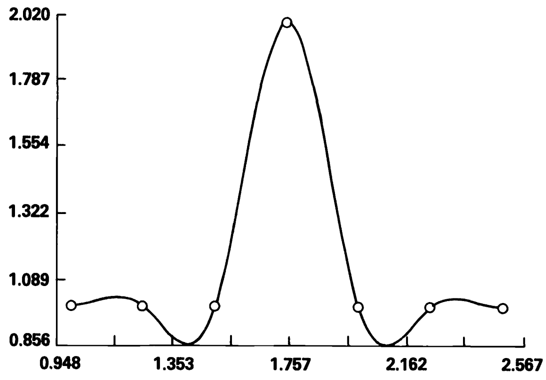


(c) Exponential spline, iteration 2.

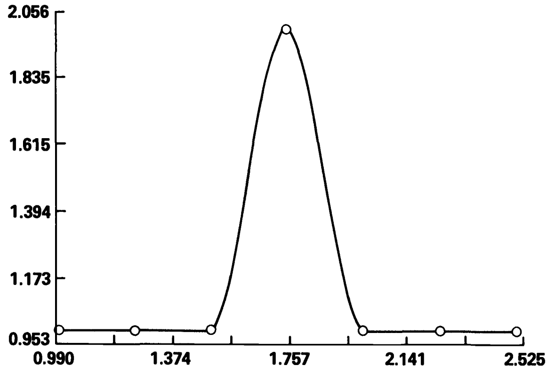


(d) Exponential spline, iteration 3.

FIG. 1. *Späth test case.*



(a) Cubic spline.



(b) Exponential spline.

FIG. 2. *Outlier test case.*

and sufficient condition for no extraneous inflection points that is the basis for our algorithm is a function of the algebraic signs of these quantities.

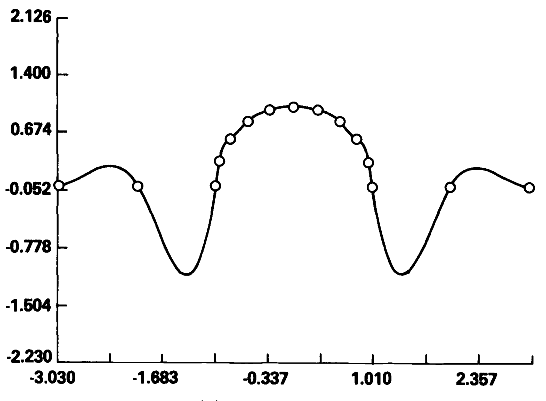
We may thus freely scale x by an arbitrary positive factor. We proceed as follows. Define $\mu = \max_{1 \leqq i \leqq N} \{p_i h_i\}$; then

$$\bar{x}_i = \frac{x_i}{a} \quad \text{with } a \geqq \frac{\mu}{\sigma} \Rightarrow p_i \bar{h}_i = \frac{1}{a} p_i h_i \leqq \frac{\mu}{a} \leqq \frac{\mu}{\mu/\sigma} = \sigma,$$

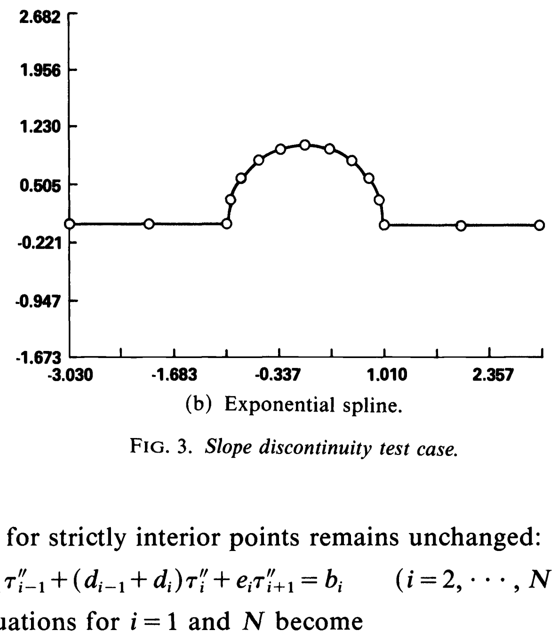
and we consequently remain within the desired number range. Whenever the p's are updated, a scaling should be done if $\mu > \sigma$. Note that when scaling is done then $f'(a)$ and $f'(b)$ should be multiplied by α.

8.3. Invariance under linear transformations. As presently proposed, the exponential spline interpolant is not invariant under change of physical units. To ensure such invariance some sort of normalization must be performed on the input data. Let $\bar{f}_i = k_1 f_i + k_2$; then $\bar{b}_i = k_1 b_i$, which $\Rightarrow \bar{\tau}_i'' = k_1 \tau_i''$. Therefore $\bar{\tau}(x) = k_1 \tau(x) + k_2$, and scaling of the ordinates is unnecessary. Let $\bar{x} = k_1 x + k_2$; then k_2 is innocuous. However, k_1 has a nonlinear effect on the interpolant. Hence, scaling of the abscissae is required. We follow Cline [4] and set $k_1 = N/\sum_i h_i$.

8.4. Periodic exponential spline. If we wish to parametrically fit a closed body in a smooth fashion, then periodic end conditions must be imposed. The necessary modifications to the preceding analysis are presented herein.



(a) Cubic spline.



(b) Exponential spline.

FIG. 3. *Slope discontinuity test case.*

The equation for strictly interior points remains unchanged:

$$e_{i-1}\tau''_{i-1}+(d_{i-1}+d_i)\tau''_i+e_i\tau''_{i+1}=b_i \qquad (i=2,\cdots,N-1).$$

While the equations for $i=1$ and N become

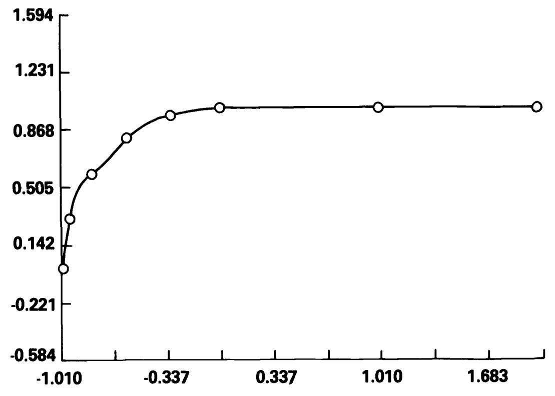
$$(d_N+d_1)\tau''_1+e_1\tau''_2+e_N\tau''_N=\frac{f_2-f_1}{h_1}-\frac{f_{N+1}-f_N}{h_N},$$

$$e_N\tau''_1+e_{N-1}\tau''_{N-1}+(d_{N+1}+d_N)\tau''_N=b_N.$$

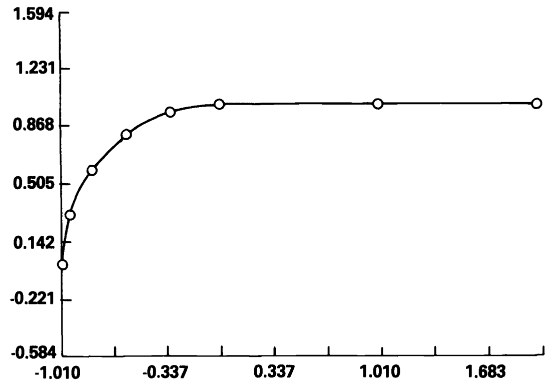
Note that the spline matrix will no longer be tridiagonal since the upper right-hand and lower left-hand corners now have nonzero entries. The Thomas algorithm is easily modified to accommodate a system of this form [2].

We treat tension parameter selection in the periodic case as follows: (1) All points $\{(x_i,f_i)\}_{i=1}^N$ are treated as interior points. (2) When $\tau''_1 b_1<0$, we modify p_1 and p_N. The previous scheme is otherwise unaltered.

8.5. Parametric exponential spline. In many applications f is not a single-valued function of x for the entire range of the data. Hence, it becomes desirable to fit both x and f versus some parameter, e.g., chordal length or arclength. Fitting versus arclength requires an additional nonlinear iteration.



(a) Cubic spline.



(b) Exponential spline.

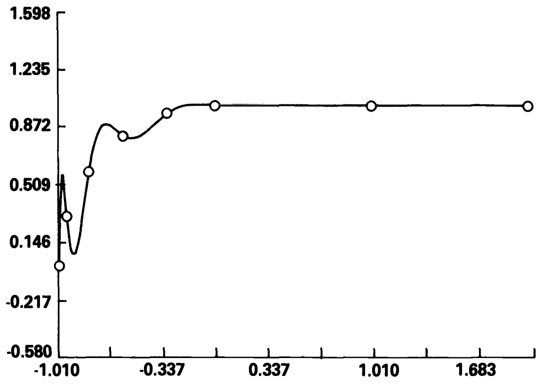
FIG. 4. *Curvature discontinuity test case.*

9. Examples. We now present a sequence of examples chosen to illustrate both the efficiency of the new parameter selection algorithm and the inherent superiority to cubic spline interpolation.

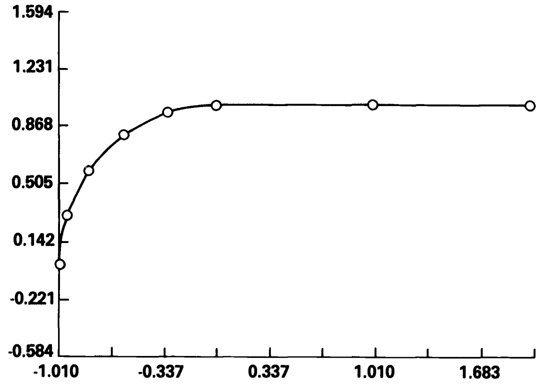
The first test case is taken from Späth's original paper [16]. The cubic spline interpolant (Fig. 1(a)) exhibits extraneous inflection points in the first, third, fourth, and eighth intervals. The converged exponential spline interpolant (Fig. 1(d)) is seen to be free of such aberrations. The general behavior of our parameter selection scheme is amply portrayed in this example: The first iteration (Fig. 1(b)) captures the gross features while subsequent iterations (Figs. 1(c)-1(d)) essentially "fine-tune" the first. We note that the scheme proposed by Späth required 12 iterations as opposed to our three iterations with no visible difference in final interpolants.

The second test case is a unit impulse function. Note the "wiggles" present in the cubic spline interpolant (Fig. 2(a)). This example demonstrates the insensitivity to "outliers" that the exponential spline interpolant (Fig. 2(b)) possesses.

The third test case is a semicircle joined to two straight line segments in such a way as to produce discontinuities in the first derivative. This example begins to implicate the cubic spline interpolant (Fig. 3(a)) as being deficient as a means of geometric representation. The exponential spline (Fig. 3(b)), on the other hand, performs ideally in this instance.



(a) Cubic spline.



(b) Exponential spline.

FIG. 5. *Vertical tangent test case.*

The fourth test case is a quarter circle joined by a straight line segment with a discontinuous second derivative at their junction. Once again, the cubic spline interpolant, (Fig. 4(a)) falls far short of geometric requirements while the exponential spline (Fig. 4(b)) does not falter.

The fifth test case displays the critical sensitivity of the cubic spline interpolant, (Fig. 5(a)) to the end conditions imposed. It is an additional advantage of the exponential spline interpolant (Fig. 5(b)) that it automatically compensates for poor end conditions, thus restricting them to local influence.

10. Conclusion. In this study, we have focused on those facets of exponential splines that pertain to their numerical computation. Spurred on by Pruess' results on the shape preservation capabilities of exponential splines [10], [11], we have devised practical tension parameter selection algorithms for co-convex and co-monotone interpolation. The end result is that the principal impediment to the widespread use of the exponential spline has been eliminated. The way is now paved for the exponential spline to supersede the cubic spline. This is natural since the cubic spline is the zeroth iterate in our procedure. Thus, if the cubic spline provides a faithful representation of the data, then the procedure will halt, whereas if the cubic spline violates the convexity and/or the monotonicity of the data then sufficient tension will automatically be employed to rectify the situation.

A broad spectrum of geometric applications in computational fluid dynamics has been described in [8]. Additionally, the use of exponential splines to approximate the solutions of systems of nonlinear hyperbolic conservation laws will be described in a forthcoming paper co-authored with A. Jameson.

Appendix A. Algorithm COCONVEX(ε, ω).

```
do  $i \leftarrow 1(1)N$
    $\dot{p}_i \leftarrow p_i$
end do
if  $\tau_1''b_1 < 0$ then
    $\bar{\lambda} \leftarrow \frac{\max(|b_1|, d_1|\tau_1''|)}{|\tau_2''|}$;  $\bar{p} \leftarrow \max((\bar{\lambda}h_1)^{-1/2}, p_1)$;  $\dot{p}_1 \leftarrow \max(\bar{p}, \dot{p}_1)$
end if
do  $i \leftarrow 2(1)N$
        if  $\tau_i'' = 0$ then
            $\dot{p}_{i-1} \leftarrow \max(p_{i-1} + \epsilon, \dot{p}_{i-1})$
        end if
        if  $\tau_i'' b_i < 0$
            $\bar{\lambda} \leftarrow \frac{\max(|b_i|, (d_{i-1} + d_i)|\tau_i''|)}{2\max(|\tau_{i-1}''|, |\tau_{i+1}''|)}$
            $\bar{p} \leftarrow \max((\bar{\lambda}h_{i-1})^{-1/2}, p_{i-1})$;     $\dot{p}_{i-1} \leftarrow \max(\bar{p}, \dot{p}_{i-1})$
            $\bar{p} \leftarrow \max((\bar{\lambda}h_i)^{-1/2}, p_i)$;         $\dot{p}_i \leftarrow \max(\bar{p}, \dot{p}_i)$
        end if
end do
if  $\tau_{N+1}'' \; b_{N+1} < 0$ then
    $\bar{\lambda} \leftarrow \frac{\max(|b_{N+1}|, d_N|\tau_{N+1}''|)}{|\tau_N''|}$;  $\bar{p} \leftarrow \max((\bar{\lambda}h_N)^{-1/2}, p_N)$;  $\dot{p}_N \leftarrow \max(\bar{p}, \dot{p}_N)$
end if
do  $i \leftarrow 1(1)N$
    $p_i \leftarrow p_i + \omega(\dot{p}_i - p_i)$
end do
```

Appendix B. Algorithm COMONOTONE(δ, ω).

```
do   $i \leftarrow 1(1)N$
     $\dot{p}_i \leftarrow p_i$
end do
do   $i \leftarrow 2(1)N - 1$
         if $m_{i-1}m_i \geq 0$ and $m_i \cdot m_{i+1} \geq 0$ then
                 if $\tau_i' \cdot m_i < 0$ then
```

```
        if    $p_{i-1} = 0$ then
              $\tilde{p}_{i-1} \leftarrow \delta$
        else
              $\bar{p} \leftarrow \frac{1 + p_{i-1}h_{i-1}}{p_{i-1}h_{i-1}} \cdot \frac{2\max(|\tau''_{i-1}|, |\tau''_i|, |\tau''_{i+1}|)}{|m_{i-1} + m_i|} + \delta$
              $\tilde{p}_{i-1} \leftarrow \max(\bar{p}, \tilde{p}_{i-1})$
        end if
        if    $p_i = 0$ then
              $\tilde{p}_i \leftarrow \delta$
        else
              $\bar{p} \leftarrow \frac{1 + p_i h_i}{p_i h_i} \cdot \frac{2\max(|\tau''_{i-1}|, |\tau''_i|, |\tau''_{i+1}|)}{|m_{i-1} + m_i|} + \delta$
              $\tilde{p}_i \leftarrow \max(\bar{p}, \tilde{p}_i)$
        end if
end if
if $\tau'_{i+1} \cdot m_{i+1} < 0$ then
        if    $p_i = 0$ then
              $\tilde{p}_i \leftarrow \delta$
        else
              $\bar{p} \leftarrow \frac{1 + p_i h_i}{p_i h_i} \cdot \frac{2\max(|\tau''_i|, |\tau''_{i+1}|, |\tau''_{i+2}|)}{|m_i + m_{i+1}|} + \delta$
              $\tilde{p}_i \leftarrow \max(\bar{p}, \tilde{p}_i)$
        end if
        if    $p_{i+1} = 0$ then
              $\tilde{p}_{i+1} \leftarrow \delta$
        else
              $\bar{p} \leftarrow \frac{1 + p_{i+1}h_{i+1}}{p_{i+1}h_{i+1}} \cdot \frac{2\max(|\tau''_i|, |\tau''_{i+1}|, |\tau''_{i+2}|)}{|m_i + m_{i+1}|}$
              $\tilde{p}_{i+1} \leftarrow \max(\bar{p}, \tilde{p}_{i+1})$
        end if
        end if
        if $\tau'_i \cdot m_i \geq 0$ and $\tau'_{i+1} m_{i+1} \geq 0$ then
                if $C_i > 0$ and $D_i < 0$ then
                        if    $m_i \leq 0$ then
                              $\tilde{p}_i \leftarrow \max(p_i, \tilde{p}_i)$
                        else
                              $\bar{p} \leftarrow -B_i/(2\sqrt{-C_i D_i}) + \delta$
                              $\tilde{p}_i \leftarrow \max(\bar{p}, p_i)$
                end if
            end if
            if $C_i < 0$ and $D_i > 0$ then
                if $m_i \geq 0$ then
                   $\hat{p}_i \leftarrow \max(p_i, \hat{p}_i)$
                else
                   $\bar{p} \leftarrow B_i/(2\sqrt{-C_iD_i}) + \delta$
                   $\hat{p}_i \leftarrow \max(\bar{p}, \hat{p}_i)$
                end if
            end if
        end if
    end if
end do
do  $i \leftarrow 1(1)N$
  $p_i \leftarrow p_i + \omega(\hat{p}_i - p_i)$
end do
```

Acknowledgments. This research was performed while the author was a graduate student at the Courant Institute of Mathematical Sciences, New York University. Consequently, it has benefited from the encouragement and advice of Charlotte Ansted, Paul Garabedian, Peter Lax, Olof Widlund, and most notably Antony Jameson.

REFERENCES

[1] M. Abramowitz and I. Stegun, *Handbook of Mathematical Functions*, National Bureau of Standards, Washington, DC, 1964.
[2] J. H. Ahlberg, E. N. Nilson, and J. L. Walsh, *The Theory of Splines and Their Applications*, Academic Press, New York, 1967.
[3] D. E. Amos, *Computation with splines and* B-*splines*, SAND78-1968, Sandia National Laboratories, Albuquerque, NM, March 1979.
[4] A. K. Cline, *Scalar- and planar-valued curve fitting using splines under tension*, Comm. ACM, 17 (1974), pp. 218-220.
[5] T. N. E. Greville, *Spline functions, interpolation and numerical quadrature*, in Mathematical Methods for Digital Computers: Vol. II, A. Ralston and H. S. Wilf, eds., John Wiley, New York, 1967, pp. 156-168.
[6] D. W. Hill, *Estimation of probability functions using splines*, Ph.D. thesis, University of New Mexico, Albuquerque, NM, 1973.
[7] D. Kershaw, *A note on the convergence of interpolatory cubic splines*, SIAM J. Numer. Anal., 8 (1971), pp. 67-74.
[8] B. J. McCartin, *Applications of exponential splines in computational fluid dynammics*, AIAA J., 21 (1983), pp. 1059-1065.
[9] ———, *Theory of exponential splines*, J. Approx. Theory, to appear.
[10] S. Pruess, *Properties of splines in tension*, J. Approx. Theory, 17 (1976), pp. 86-96.
[11] ———, *Alternatives to the exponential spline in tension*, Math. Comp., 33 (1979), pp. 1273-1281.
[12] P. Rentrop, *An algorithm for the computation of the exponential spline*, Numer. Math., 35 (1980), pp. 81-93.
[13] I. J. Schoenberg, *Cardinal Spline Interpolation*, CBMS-NSF Region Conference Series in Applied Mathematics 12, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1973.

[14] D. G. SCHWEIKERT, *An interpolation curve using a spline in tension*, J. Math. Phys., 45 (1966), pp. 312-317.

[15] T. I. SEIDMAN AND R. J. KORSAN, *Endpoint formulas for interpolatory cubic splines*, Math. Comp., 26, (1972), pp. 879-900.

[16] H. SPÄTH, *Exponential spline interpolation*, Computing, 4 (1969), pp. 225-233.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 2, pp. 263-280, March 1990

A MATRIX PROBLEM WITH APPLICATION TO RAPID SOLUTION OF INTEGRAL EQUATIONS*

LOTHAR REICHEL†

Abstract. Let $\{z_j\}_{j=0}^{n-1}$ be a set of distinct points in the complex plane $\mathbb{C}$, and introduce the $n \times n$ matrix $A_n = [a_{jk}]_{j,k=0}^{n-1}$, $a_{jk} = (z_j - z_k)^{-1}$, $j \neq k$ and $a_{jj} = 0$. Recently Golub and Trummer raised the question of whether or not, for an arbitrary vector $x \in \mathbb{C}^n$, Ax can be computed in fewer than $O(n^2)$ arithmetic operations by using the structure of A_n. In this paper it is assumed that there is a smooth 2π-periodic bijective function $z(t)$ such that $z_j = z(2\pi(j-1)/n)$, $j = 1(1)n$, and shown that when n increases, there is a sequence of matrices of low rank $\tilde{A}_n$, $n = 1, 2, 3, \cdots$ such that $\tilde{A}_n \to A_n$ as $n \to \infty$ and $\tilde{A}_n x$ can be computed in $O(n \log n)$ arithmetic operations. The method to construct the matrices $\tilde{A}_n$ is then used in a fast solution scheme for Fredholm integral equations of the second kind with smooth periodic kernels. The integral equations are discretized by the trapezoidal rule using the nodes $z_j = z(2\pi j/n)$, $0 \leqq j < n$, and it is shown that arbitrarily accurate approximate solutions can be computed in $O(n \log n)$ arithmetic operations for large n, provided that $z(t)$ is sufficiently smooth. When the asymptotic analysis is not applicable, fast iterative $O(n^2)$ solution methods are obtained. The scheme is applied to the solution of a Fredholm integral equation of the second kind of plane potential theory and Cauchy singular integral equations.

Key words. integral equation, fast solution of linear systems, matrix vector multiplication

AMS(MOS) subject classifications. 65R20, 65F05, 65F10, 65F30

1. Introduction.

Let $\{z_j\}_{j=0}^{n-1}$ be a set of n distinct points in the complex plane $\mathbb{C}$, and consider the complex skew-symmetric matrix $A_n = [a_{jk}]_{j,k=0}^{n-1}$ defined by

$$a_{jk} := \begin{cases} (z_j - z_k)^{-1}, & j \neq k, \\ 0, & j = k. \end{cases} \tag{1.1}$$

Golub and Trummer [GT] recently raised the question of whether or not, for an arbitrary vector $\mathbf{x} \in \mathbb{C}^n$, $A_n\mathbf{x}$ can be computed in less than than $O(n^2)$ arithmetic operations by using the structure of the matrix. We note that if the z_j are equidistant points on a circle and $D_n := \operatorname{diag}(z_0, z_1, \cdots, z_{n-1})$, then $A_n D_n$ is a Hermitian circulant matrix, and $A_n\mathbf{x}$ can be determined in $O(n \log n)$ arithmetic operations. To see this, we introduce the unitary Fourier matrix $W_n = [w_{jk}]$:

$$\begin{aligned} w_{2j,k} &:= n^{-1/2} \exp(-2\pi ijk/n), \qquad 0 \leqq j \leqq n_1, \quad 0 \leqq k < n, \\ w_{2j-1,k} &:= n^{-1/2} \exp(2\pi ijk/n), \qquad 1 \leqq j \leqq n_2, \quad 0 \leqq k < n, \end{aligned} \tag{1.2}$$

where $i := \sqrt{-1}$, n_1 is the integer part of $(n-1)/2$, and n_2 is the integer part of $n/2$. The fast Fourier transform (FFT) method allows us to compute $W_n\mathbf{y}$ and $W_n^H\mathbf{y}$ in $O(n \log n)$ arithmetic operations for any $\mathbf{y} \in \mathbb{C}^n$ and for any n [He, Thm. 13.7e]. Moreover, for any $n \times n$ circulant matrix C, $W_n C W_n^H$ is a diagonal matrix whose elements can be computed in $O(n \log n)$ arithmetic operations [Da, Thm. 3.2.2]. For future reference, we note that the elements of the matrix

$$D_n^{(2)} := \operatorname{diag}(d_0^{(2)}, d_1^{(2)}, \cdots, d_{n-1}^{(2)}) := W_n A_n D_n W_n^H \tag{1.3}$$

* Received by the editors May 13, 1987; accepted for publication (in revised form) January 18, 1989.

† University of Kentucky, Department of Mathematics, University of Lexington, Kentucky 40506. Bergen Scientific Centre, Allégaten 36, N-5007 Bergen, Norway.

can be determined explicitly. For n even, we obtain

$$(1.4)\qquad \begin{aligned} d_{2j}^{(2)} &= j-\frac{n-1}{2}, \qquad 0 \leqq j < \frac{n}{2}, \\ d_{2j-1}^{(2)} &= \frac{n+1}{2}-j, \qquad 1 \leqq j \leqq \frac{n}{2} \end{aligned}$$

(see Appendix for details). From (1.3) and $W_n^H W_n = I$ we get

$$(1.5)\qquad A_n\mathbf{x} = W_n^H D_n^{(2)} W_n D_n^{-1}\mathbf{x}.$$

The right-hand side of (1.5) can be computed in $O(n \log n)$ arithmetic operations.

We next indicate how approximations of $A_n\mathbf{x}$ can be determined for a more general choice of z_j. Let Γ be a smooth Jordan curve with parametric representation $z(t)$, $0 \leqq t < 2\pi$, and define $z(2\pi) := z(0)$. Assume that there is a constant δ with

$$(1.6)\qquad |z'(t)| \geqq \delta > 0, \qquad 0 \leqq t \leqq 2\pi.$$

Let D_n henceforth be the diagonal matrix

$$(1.7)\qquad D_n := i \operatorname{diag}(z'(0), z'(2\pi/n), z'(4\pi/n), \cdots, z'(2\pi(n-1)/n)).$$

Assume that $z_j := z(2\pi j/n)$ for $0 \leqq j < n$. We show in § 2 that if $z(t)$ is sufficiently smooth and n is sufficiently large, then the matrix A_nD_n can be split according to

$$A_nD_n = B_n - B_n^{(0)} - C_n,$$

where C_n is a circulant matrix, $B_n^{(0)}$ is a diagonal matrix, and the matrix B_n can be approximated well by a matrix $B_n^{(2)}$ of low rank.

We determine a representation of $B_n^{(2)}$ by the FFT-method in $O(n \log n)$ arithmetic operations and define $A_n^{(2)} := (B_n^{(2)} - B_n^{(0)} - C_n)D_n^{-1}$. The representation of $B_n^{(2)}$ is such that $A_n^{(2)}\mathbf{x}$ can be computed in $O(n \log n)$ arithmetic operations by the FFT method as $n \to \infty$. Moreover, $A_n^{(2)} \to A_n$ as $n \to \infty$, where we show convergence in the least squares norm and in the uniform norm for sufficiently smooth $z(t)$.

A different solution of the Golub-Trummer problem has been presented by Gerasoulis, Grigoriadis, and Sun [GGS] who show that $A\mathbf{x}$ can be computed in $O(n \log^2 n)$ arithmetic operations as n increases for *any point set* $\{z_j\}_{j=0}^{n-1}$ of n distinct points if exact arithmetic is used. For general point sets the determination of $A\mathbf{x}$ by the method of [GGS] involves computations that are unstable.

In § 3 the approximation of A_nD_n by $B_n^{(2)} - B_n^{(0)} - C_n$ is applied to devise rapid schemes for solving Fredholm integral equations of the second kind with smooth periodic kernels. We illustrate the schemes by considering the integral equation

$$(1.8)\qquad \frac{1}{2}\sigma(z) + \operatorname{Re}\left(\frac{1}{2\pi i}\int_\Gamma \frac{1}{\zeta - z}\sigma(\zeta)\, d\zeta\right) = f(z), \quad z \in \Gamma$$

for σ. We discretize (1.8) by the Nyström method based on the trapezoidal rule, and the matrix obtained is closely related to A_n. Approximating the matrix obtained from (1.8) by a low rank approximation, similarly as A_n is approximated by $A_n^{(2)}$ in § 2, yields a linear system of equations that can be computed and solved in only $O(n \log n)$ arithmetic operations for n large. This modified system has a solution that converges to the solution of the linear system of equations obtained by discretizing (1.8) as $n \to \infty$, provided that Γ is sufficiently smooth.

Fredholm integral equations of the first kind with a logarithmic kernel behave essentially as Fredholm integral equations of the second kind (see [LSW], [Be], [Re]). Our $O(n \log n)$ solution method can also be used to solve boundary value problems for the Laplace equation formulated with these integral equations.

If n is not large enough so that the matrix obtained from (1.8) can be replaced by a low-rank approximation, then we use the low-rank approximation to obtain $O(n^2)$ iterative schemes, whose rate of convergence increases with n, a behavior that had been established previously for multigrid methods [Ha], [Sch]. The iterative methods considered are a block Jacobi method and a preconditioned conjugate gradient method. Other iterative methods for Fredholm integral equations of the second kind have recently been discussed in [At], [DF], [Ha], [HS], [Ro], and [GR]. Section 4 contains computed examples of our solution method applied to (1.8).

We remark that the solution method by Gerasoulis of the Golub-Trummer matrix problem also can be applied to the solution of integral equations, more precisely to the rapid solution of Cauchy singular integral equations on an interval [Ge].

2. The matrix problem. The analysis of this section is carried out under the assumption that $z(t) \in C^{m+2,\alpha}[0, 2\pi]$, i.e., $d^{m+2}z/dt^{m+2}$ exists and satisfies a Hölder condition with Hölder constant $\alpha, 0<\alpha<1$. Requirements on m and α are specified below. We also assume $z(t)$ satisfies (1.6). Let A_n be defined by (1.1) with $z_j := z(2\pi j/n)$, $0 \leqq j < n$, and let D_n be defined by (1.7). We introduce the function

$$(2.1)\qquad \begin{aligned} b(s,t) &:= \frac{e^{it}}{e^{is}-e^{it}} + i\frac{z'(t)}{z(s)-z(t)}, \qquad s \neq t, \\ b(t,t) &:= -\frac{1}{2}\left(1 + i\frac{z''(t)}{z'(t)}\right) \end{aligned}$$

and the matrices $B_n = [b_{jk}]$,

$$(2.2)\qquad b_{jk} := b\left(\frac{2\pi j}{n}, \frac{2\pi k}{n}\right), \qquad 0 \leqq j, k < n,$$

$$(2.3)\qquad \begin{aligned} B_n^{(0)} &:= \operatorname{diag}(b_{00}, b_{11}, \cdots, b_{n-1,n-1}), \\ C_n &:= B_n - B_n^{(0)} - A_n D_n. \end{aligned}$$

C_n is the Hermitian circulant matrix with $W_n C_n W_n^H = D_n^{(2)}$, where $D_n^{(2)}$ is defined by (1.3). We now derive a low-rank approximation $B_n^{(2)}$ of B_n. This requires estimates for certain Fourier coefficients. By Taylor's theorem with integral remainder, it follows that $s \to b(s,t) \in C^{m,\alpha}[0, 2\pi]$ uniformly for $0 \leqq t \leqq 2\pi$, and $t \to b(s,t) \in C^{m,\alpha}[0,2\pi]$ uniformly for $0 \leqq s \leqq 2\pi$. Therefore the Fourier coefficients

$$(2.4)\qquad g_{jk} := \frac{1}{4\pi^2}\int_0^{2\pi}\int_0^{2\pi} b(s,t)\, e^{ijs}\, e^{ikt}\, ds\, dt$$

satisfy for some constant c depending on $z(t)$ but independent of j, k,

$$(2.5)\qquad |g_{jk}| \leqq c(\max\{|j|+1, |k|+1\})^{-m-\alpha} \quad \forall j, k$$

(see [Ka, Chap. 1]). If $z(t)$ is analytic, then there are constants c and ρ, such that $0 \leqq \rho < 1$ and $|g_{jk}| \leqq c\rho^{j+k}$ for all j and k (see [He, Chap. 13]). Define the matrix $G^{(2)} = [g_{jk}^{(2)}]$ by

$$(2.6)\qquad G_n^{(2)} = n^{-1} W_n B_n W_n^H.$$

The $g_{jk}^{(2)}$ are discrete Fourier coefficients of $b(s,t)$. In order to bound the $g_{jk}^{(2)}$, we introduce

$$g_{jk}^{(1)} := \frac{1}{n^2}\sum_{p=0}^{n-1}\sum_{q=0}^{n-1} b\left(\frac{2\pi p}{n}, \frac{2\pi q}{n}\right) e^{2\pi i p j/n}\, e^{2\pi i q k/n},$$

$$g_k^{(1)}(s) := \frac{1}{n}\sum_{q=0}^{n-1} b\left(s, \frac{2\pi q}{n}\right) e^{2\pi i q k/n},$$

$$g_k(s) := \frac{1}{2\pi}\int_0^{2\pi} b(s,t)\, e^{ikt}\, dt.$$

Since $t \to b(s,t) \in C^{m+\alpha}[0, 2\pi]$, uniformly for $-1 \leqq s \leqq 1$, there is a constant c' independent of k and s, such that

$$|g_k(s)| \leqq c'(|k|+1)^{-m-\alpha}, \qquad -1 \leqq s \leqq 1 \quad \text{for all } k. \tag{2.7}$$

Assume that $m+\alpha > 1$, and substitute (2.7) into

$$g_k^{(1)}(s) - g_k(s) = \sum_{p=1}^{\infty} (g_{k+pn}(s) + g_{k-pn}(s)),$$

in order to obtain

$$|g_k^{(1)}(s) - g_k(s)| \leqq c' \sum_{p=1}^{\infty} ((pn+k)^{-m-\alpha} + (pn-k)^{-m-\alpha}).$$

Hence, there is a constant c'', independent of s and k, such that

$$|g_k^{(1)}(s) - g_k(s)| \leqq c'' n^{-m-\alpha}, \quad -1 \leqq s \leqq 1, \qquad -n/2 \leqq k \leqq n/2.$$

Therefore

$$\begin{aligned}
|g_{jk}^{(1)} - g_{jk}| &= \left|\frac{1}{n}\sum_{p=0}^{n-1} g_k^{(1)}\left(\frac{2\pi p}{n}\right) e^{2\pi i p j/n} - g_{jk}\right| \\
&\leqq \left|\frac{1}{n}\sum_{p=0}^{n-1}\left(g_k^{(1)}\left(\frac{2\pi p}{n}\right) - g_k\left(\frac{2\pi p}{n}\right)\right) e^{2\pi i p j/n}\right| + \left|\frac{1}{n}\sum_{p=0}^{n-1} g_k\left(\frac{2\pi p}{n}\right) e^{2\pi i p j/n} - g_{jk}\right| \\
&\leqq c'' n^{-m-\alpha} + \left|\sum_{p=1}^{\infty}(g_{j+pn,k} + g_{j-pn,k})\right| \leqq c''' n^{-m-\alpha},
\end{aligned}$$

where c''' is a constant independent of $-n/2 \leqq j,\ k \leqq n/2$. In particular, we have shown that

$$|g_{jk}^{(1)}| \leqq \tilde{c}(\max\{|k|+1, |j|+1\})^{-m-\alpha}, \qquad -\frac{n}{2} \leqq j,\ k \leqq \frac{n}{2}$$

for some constant $\tilde{c}$ independent of j and k. Finally, for $0 \leqq j,\ k < n/2$,

$$g_{2j,2k}^{(2)} = g_{j,-k}^{(1)}, \qquad g_{2j,2k+1}^{(2)} = g_{j,k+1}^{(1)},$$
$$g_{2j+1,2k}^{(2)} = g_{-j-1,-k}^{(1)}, \qquad g_{2j+1,2k+1}^{(2)} = g_{-j-1,k+1}^{(1)},$$

shows that

$$|g_{jk}^{(2)}| \leqq c_1(\max\{j+1, k+1\})^{-m-\alpha}, \qquad 0 \leqq j,\ k < n, \tag{2.8}$$

provided that $m+\alpha > 1$. This is assumed henceforth. From (2.8) it follows that $G_n^{(2)}$ can be approximated well by a matrix of low rank. From this low-rank matrix we will obtain $B_n^{(2)}$, a computationally suitable low-rank approximant of B_n.

A low-rank approximant of $G_n^{(2)}$ is introduced in two steps. First we define for some integer $l > 0$, to be specified below, the matrix $G_n^{(3)} = [g_{jk}^{(3)}]_{j,k=0}^{n-1}$ defined by

$$g_{jk}^{(3)} := \begin{cases} g_{jk}^{(2)}, & 0 \leqq j,\ k < l, \\ 0, & l \leqq j < n \text{ or } l \leqq k < n. \end{cases} \tag{2.9}$$

For $l \geqq 1$ and $m+\alpha > \frac{1}{2}$ the inequality

$$\sum_{j=l}^{n-1} (j+1)^{-2(m+\alpha)} \leqq \int_l^{\infty} x^{2(m+\alpha)} = \frac{l^{-2(m+\alpha)+1}}{2(m+\alpha)-1} \tag{2.10}$$

holds. From (2.10) and similar inequalities, we obtain

$$\|G_n^{(2)}-G_n^{(3)}\|_2 \leqq \|G_n^{(2)}-G_n^{(3)}\|_F \leqq cl^{-m-\alpha+1}, \tag{2.11a}$$

$$\|G_n^{(2)}-G_n^{(3)}\|_\infty \leqq cl^{-m-\alpha+1} \tag{2.11b}$$

where $\| \ \|_2$ denotes the L_2-norm, $\| \ \|_F$ the Frobenius norm, $\| \ \|_\infty$ the uniform norm, and c is a constant depending on m and α but independent of l and n.

Let B_l be the matrix obtained by tabulating $b(s,t)$ at l^2 points (see (2.2)), and let W_l be defined by (1.2). Analogously to (2.6) the elements of $G_l^{(4)}=[g_{jk}^{(4)}]_{j,k=0}^{l-1}$,

$$G_l^{(4)} := l^{-1} W_l B_l W_l^H \tag{2.12}$$

are discrete Fourier coefficients of $b(s,t)$. By (2.5) and (2.7) there is a constant c' independent of j, k, l, n such that

$$|g_{jk}^{(4)}-g_{jk}^{(3)}| \leqq c' l^{-m-\alpha}, \qquad 0 \leqq j, k < l, \tag{2.13}$$

provided that $m+\alpha>1$.

Let $G_n^{(5)}=[g_{jk}^{(5)}]_{j,k=0}^{n-1}$ be defined by

$$g_{jk}^{(5)} = \begin{cases} g_{jk}^{(4)}, & 0 \leqq j, k < l, \\ 0 & l \leqq j < n \text{ or } l \leqq k < n. \end{cases} \tag{2.14}$$

By (2.13),

$$\|G_n^{(5)}-G_n^{(3)}\|_2 \leqq \|G_n^{(5)}-G_n^{(3)}\|_F \leqq c' \, l^{-m-\alpha+1}, \tag{2.15a}$$

$$\|G_n^{(5)}-G_n^{(3)}\|_\infty \leqq c' l^{-m-\alpha+1}. \tag{2.15b}$$

We are now in a position to describe the computation of an approximation of $A_n\mathbf{x}$. We introduce the low-rank approximation

$$B_n^{(2)} := nW_n^H G_n^{(5)} W_n \tag{2.16}$$

of B_n, and approximate A_n by

$$A_n^{(2)} = (B_n^{(2)} - B_n^{(0)} - C_n)D_n^{-1} = (W_n^H(nG_n^{(5)} - D_n^{(2)})W_n - B_n^{(0)})D_n^{-1}. \tag{2.17}$$

Then

$$\begin{aligned} A_n - A_n^{(2)} = (B_n - B_n^{(2)})D_n^{-1} &= nW_n^H(G_n^{(2)}-G_n^{(3)})W_nD_n^{-1} \\ &+ nW_n^H(G_n^{(3)}-G_n^{(5)})W_nD_n^{-1}. \end{aligned} \tag{2.18}$$

By (2.18) we obtain using $W_n^H W_n = I$, (1.5), (2.11), and (2.15), that for some constant c independent of l and n,

$$\begin{aligned} \|nW_n^H(G_n^{(2)}-G_n^{(3)})W_nD_n^{-1}\|_2 &\leqq n\|G_n^{(2)}-G_n^{(3)}\|_2\|D_n^{-1}\|_2 \leqq cnl^{-m-\alpha+1}, \\ \|nW_n^H(G_n^{(3)}-G_n^{(5)})W_nD_n^{-1}\|_2 &\leqq n\|G_n^{(3)}-G_n^{(5)}\|_2\|D_n^{-1}\|_2 \leqq cnl^{-m-\alpha+1}. \end{aligned} \tag{2.19a}$$

Since $\|W_n\|_\infty := \|W_n^H\|_\infty = \sqrt{n}$, we obtain

$$\begin{aligned} \|nW_n^H(G_n^{(2)}-G_n^{(3)})W_nD_n^{-1}\|_\infty &\leqq cn^2 l^{-m-\alpha+1}, \\ \|nW_n^H(G_n^{(3)}-G_n^{(5)})W_nD_n^{-1}\|_\infty &\leqq cn^2 l^{-m-\alpha+1}. \end{aligned} \tag{2.19b}$$

For some arbitrary but fixed constants $0<d_1<d_2<\infty$, let $l=l(n)$ be such that $d_1 n^{1/2} \leqq l \leqq d_2 n^{1/2}$. By (2.19) we obtain

$$\|A_n - A_n^{(2)}\|_2 = O(n^{(3-m-\alpha)/2}), \qquad n \to \infty, \tag{2.20a}$$

i.e., $m+\alpha>3$ is sufficient for convergence. Similarly,

$$\|A_n - A_n^{(2)}\|_\infty = O(n^{(5-m-\alpha)/2}), \qquad n \to \infty, \tag{2.20b}$$

and $m+\alpha>5$ suffices for convergence. For any $\mathbf{y}\in\mathbb{C}^n$, $G^{(5)}\mathbf{y}$ can be computed in $O(l^2)=O(n)$ arithmetic operations. Therefore $A_n^{(2)}\mathbf{x}$ can be computed in $O(n\log n)$ arithmetic operations using the right-hand side of (2.17); see also Example 2.1 below. We note that convergence in (2.20) can also be shown for different choices of increasing functions $l(n)$. Hence, arbitrarily good approximations of $A_n\mathbf{x}$ can be computed in $O(n\log n)$ operations as $n\to\infty$ provided that the z_j lie on a sufficiently smooth curve.

Example 2.1. We compare the number of complex multiplications required for evaluating $A_n\mathbf{x}$ and $A_n^{(2)}\mathbf{x}$ for $\mathbf{x}\in\mathbb{C}^n$. Let $\mu_{\mathbb{C}}$ denote complex multiplication. For $n=2^p$ with $p\geqq 0$ an integer, the vectors $n^{1/2}W_n\mathbf{x}$ and $n^{1/2}W_n^H\mathbf{x}$ can be computed by $\frac{1}{2}n\log_2 n\,\mu_{\mathbb{C}}$ (see [He, Thm. 13.1b]). We compute $A_n^{(2)}\mathbf{x}$ from the following formula, very similar to (2.18):

$$(2.19)\qquad A_n^{(2)}\mathbf{x}=((n^{1/2}W_n^H)(G_n^{(5)}-n^{-1}D_n^{(2)})(n^{1/2}W_n)-B_n^{(0)})D_n^{-1}\mathbf{x}.$$

Table 2.1 yields the operation count for computing $A_n^{(2)}\mathbf{x}$ by the right-hand side of (2.19). A complex division is counted as $1.5\,\mu_{\mathbb{C}}$, and a real multiplication counts as $0.25\,\mu_{\mathbb{C}}$. If n or l were not a power of two the count of $\mu_{\mathbb{C}}$ in Table 2.1 would increase. An upper bound for the number of $\mu_{\mathbb{C}}$ required for any $n\geqq 0$ and $0\leqq l\leqq n$ can be determined by using that $n^{1/2}W_n\mathbf{x}$ and $n^{1/2}W_n^H\mathbf{x}$ can be computed in at most $6n\log_2 n+14n\,\mu_{\mathbb{C}}$ for any $n>0$ (see [He, Thm. 13.7e]). The evaluation of $A^{(2)}\mathbf{x}$ is seen to require fewer $\mu_{\mathbb{C}}$ than the evaluation of $A_n\mathbf{x}$ already for fairly small values of n; see Table 2.2. The convergence of $A_n^{(2)}\mathbf{x}$ to $A_n\mathbf{x}$ as n increases is illustrated in numerical examples of § 4. □

3. Solution of integral equations. Let Ω be a bounded simply connected plane region with a smooth boundary Γ, oriented so that Ω lies to the left when traversing Γ. Let $f(z)$ be a given real-valued, continuous, piecewise, smooth function on Γ. We

TABLE 2.1
Number of complex multiplications $\mu_{\mathbb{C}}$ for n and l a power of two.

	$\mu_{\mathbb{C}}$
Multiplication by D_n^{-1}	$1.5n$
Multiplication by $B_n^{(0)}$	n
Multiplication by $n^{1/2}W_n$	$0.5n\log_2 n$
Multiplication by $D_n^{(2)}$	n
Multiplication by n^{-1}	$0.25n$
Forming $G_n^{(5)}$	$0.5l^2\log_2 l$
Multiplication by $G_n^{(5)}$	l^2
Multiplication by $n^{1/2}W_n^H$	$0.5n\log_2 n$
Total number of $\mu_{\mathbb{C}}$ for $l=2n^{1/2}$	$2n\log_2 n+9.75n$

TABLE 2.2
Comparison of n^2 $\mu_{\mathbb{C}}$ for the evaluation of $A_n\mathbf{x}$ and $2n\log_2 n+9.75n$ $\mu_{\mathbb{C}}$ for the evaluation of $A^{(2)}\mathbf{x}$.

n	l	n^2	$2n\log_2 n+9.75n$
16	8	$2.6\cdot 10^2$	$2.8\cdot 10^2$
64	16	$4.1\cdot 10^3$	$1.4\cdot 10^3$
256	32	$6.6\cdot 10^4$	$6.6\cdot 10^3$
1024	64	$1.0\cdot 10^6$	$3.0\cdot 10^4$
4096	128	$1.7\cdot 10^7$	$1.4\cdot 10^5$

first consider solution methods for (1.8), a Fredholm integral equation of the second kind. The integral equation yields a solution to the Dirichlet problem

$$\Delta u = 0 \quad \text{in } \Omega, \qquad u = f \quad \text{on } \Gamma. \tag{3.1}$$

In later subsections we discuss generalizations to solutions of exterior Dirichlet problems and to the solution of Cauchy singular integral equations. It is hoped that these examples indicate how the solution method can be used for solving other integral equations as well. We wish to stress that our solution schemes are applicable to any Fredholm integral equations of the second kind with a smooth periodic kernel.

3.1. Interior Dirichlet problems. Let $z(t)$ be a parametric representation of Γ satisfying (1.6), and assume that $z(t) \in C^{m+2,\alpha}[0, 2\pi]$ for $m + \alpha > 1$. Introduce $\tilde{f}(t) := 2f(z(t))$, $\tilde{\sigma}(t) := \sigma(z(t))$, $0 \leqq t < 2\pi$. Then (1.8) becomes

$$\tilde{\sigma}(s) - \frac{1}{\pi} \operatorname{Im}\left(\int_0^{2\pi} \frac{z'(t)}{z(s) - z(t)} \tilde{\sigma}(t)\, dt\right) = \tilde{f}(s), \qquad 0 \leqq s \leqq 2\pi. \tag{3.2}$$

Analogously to § 2 we split the kernel:

$$\begin{aligned} &\tilde{\sigma}(s) - \frac{1}{\pi} \operatorname{Im}\left(\int_0^{2\pi} \frac{ie^{it}}{e^{is} - e^{it}} \tilde{\sigma}(t)\, dt\right) \\ &+ \frac{1}{\pi} \operatorname{Re}\left(\int_0^{2\pi} \left(\frac{e^{it}}{e^{is} - e^{it}} + i \frac{z'(t)}{z(s) - z(t)}\right) \tilde{\sigma}(t)\, dt\right) = \tilde{f}(s), \qquad 0 \leqq s \leqq 2\pi. \end{aligned} \tag{3.3}$$

By $\operatorname{Im}(ie^{it}/e^{is} - e^{it}) = -\frac{1}{2}$ and (2.1), we can write (3.3) as

$$\tilde{\sigma}(s) + \frac{1}{2\pi} \int_0^{2\pi} \tilde{\sigma}(t)\, dt + \frac{1}{\pi} \int_0^{2\pi} \operatorname{Re}(b(s, t)) \tilde{\sigma}(t)\, dt = \tilde{f}(s), \qquad 0 \leqq s \leqq 2\pi. \tag{3.4}$$

We discretize (3.4) by the Nyström method based on the trapezoidal rule with nodes $t_j = 2\pi j/n$, $0 \leqq j < n$, and introduce $\boldsymbol{\sigma} := (\tilde{\sigma}(t_0), \tilde{\sigma}(t_1), \cdots, \tilde{\sigma}(t_{n-1}))^T$ and $\mathbf{f} := (\tilde{f}(t_0), \tilde{f}(t_1), \cdots, \tilde{f}(t_{n-1}))^T$. We obtain the algebraic linear system of equations,

$$\left(I + C_n^{(3)} + \frac{2}{n} \operatorname{Re}(B_n)\right) \boldsymbol{\sigma} = \mathbf{f} \tag{3.5}$$

where $C_n^{(3)}$ is the $n \times n$ circulant matrix with all entries $1/n$, and where B_n is defined by (2.2). For future reference we define

$$D_n^{(3)} := W_n C_n^{(3)} W_n^H = \operatorname{diag}(1, 0, \cdots, 0). \tag{3.6}$$

We define a low-rank approximation of $\operatorname{Re}(B_n)$ similar to the way we defined a low-rank approximation of B_n in § 2. Let

$$H_n^{(2)} := W_n \left(\frac{2}{n} \operatorname{Re}(B_n)\right) W_n^H \tag{3.7}$$

and let $H_n^{(3)}$ be the $n \times n$ matrix with the same $l \times l$ principal submatrix as $H_n^{(2)}$, and all other elements zero. Since $\operatorname{Re}(b(s, t))$ is at least as smooth as $b(s, t)$, we obtain similarly to (2.11)

$$\|H_n^{(2)} - H_n^{(3)}\|_2 \leqq cl^{-m-\alpha+1}, \tag{3.8}$$

for some constants c, independent of l and n. We introduce

$$H_l^{(4)} := W_l \left(\frac{2}{l} \operatorname{Re}(B_l)\right) W_l^H, \tag{3.9}$$

and $H_n^{(5)}$, the $n\times n$ matrix with $H_l^{(4)}$ as leading principal submatrix and all other elements zero. $H_l^{(4)}$ can be computed in $O(l^2\log l)$ arithmetic operations (see [He]) where also a recursive scheme for the computation of multivariate Fourier transforms is presented. Similarly to (2.15), we obtain

$$\|H_n^{(5)}-H_n^{(3)}\|_2\leqq c'l^{-m-\alpha+1}, \tag{3.10}$$

for some constant c' independent of l and n. The low-rank approximation of $(2/n)\operatorname{Re}(B_n)$ to be used is

$$B_n^{(3)}:=W_n^H H_n^{(5)} W_n. \tag{3.11}$$

We also introduce

$$B_n^{(4)}:=\frac{2}{n}\operatorname{Re}(B_n)-B_n^{(3)}, \tag{3.12}$$

and obtain from (3.8), (3.10), and

$$B_n^{(4)}=W_n^H(H_n^{(2)}-H_n^{(3)})W_n+W_n^H(H_n^{(3)}-H_n^{(5)})W_n, \tag{3.13}$$

that for some constant c independent of n and l

$$\|B_n^{(4)}\|_2\leqq cl^{-m-\alpha+1}. \tag{3.14}$$

We write (3.5) in the form

$$(I+C_n^{(3)}+B_n^{(3)})\boldsymbol{\sigma}=-B_n^{(4)}\boldsymbol{\sigma}+\mathbf{f}. \tag{3.15}$$

LEMMA 3.1. *There are constants l_0 and c, depending on Γ, such that for each $l\geqq l_0$ and $n\geqq l$ the matrix $(I+C_n^{(3)}+B_n^{(3)})^{-1}$ exists and*

$$\|(I+C_n^{(3)}+B_n^{(3)})^{-1}\|_2\leqq c. \tag{3.16}$$

Proof. Let $\tilde{W}_n$ be the unitary $n\times n$ matrix

$$\tilde{W}_n:=\begin{pmatrix} W_l & 0\\ 0 & I\end{pmatrix}\begin{matrix}\}\ l\\ \}\ n-l\end{matrix}$$

and introduce the block diagonal matrix

$$\tilde{A}_l:=\begin{pmatrix} I+C_l^{(3)}+(2/l)\operatorname{Re}(B_l) & 0\\ 0 & I\end{pmatrix}\begin{matrix}\}\ l\\ \}\ n-l\end{matrix}.$$

Then

$$I+C_n^{(3)}+B_n^{(3)}=W_n^H(I+D_n^{(3)}+H_n^{(5)})W_n=W_n^H\tilde{W}_n\tilde{A}_l\tilde{W}_n^H W_n. \tag{3.17}$$

The integral operator of (3.2) is a compact perturbation of the identity, and (3.2) is uniquely solvable. $(I+C_l^{(3)}+(2/l)\operatorname{Re}(B_l))$ is a discretization of this integral operator such that by [At, Thm. 4, p. 97] there are constants l_0 and c' with the property that for $l\geqq l_0$ the inverse $(I+C_l^{(3)}+(2/l)\operatorname{Re}(B_l))^{-1}$ exists and $\|(I+C_l^{(3)}+(2/l)\operatorname{Re}(B_l))^{-1}\|_2\leqq c'$. Let $c:=\max\{c',1\}$. Then $\|\tilde{A}_l^{-1}\|_2\leqq c$, and (3.16) follows by (3.17). □

THEOREM 3.2. *For some constants $0<d_1<d_2<\infty$, let $l=l(n)$ be an integer such that $d_1n^{1/3}\leqq l\leqq d_2n^{1/3}$. Assume $l\geqq l_1$ for a constant l_1 defined in the proof. Let $\boldsymbol{\sigma}^*$ solve (3.5) and let $\tilde{\boldsymbol{\sigma}}$ satisfy*

$$(I+C_n^{(3)}+B_n^{(3)})\tilde{\boldsymbol{\sigma}}=\mathbf{f}. \tag{3.18}$$

Then $\tilde{\boldsymbol\sigma}$ can be computed in $O(n \log n)$ arithmetic operations as $n \to \infty$, where the coefficient for $n \log n$ is independent of d_1 and d_2. Let n_1 be a sufficiently large constant. Then there is a constant c', independent of $l \geqq l_1$ and $n \geqq n_1$, such that

$$(3.19a)\qquad \begin{aligned} &\|(I + C_n^{(3)} + B_n^{(3)} + B_n^{(4)})\tilde{\boldsymbol\sigma} - \mathbf{f}\|_2 \leqq c'\|\mathbf{f}\|_2 n^{(1-m-\alpha)/3},\\ &\|\tilde{\boldsymbol\sigma} - \boldsymbol\sigma^*\|_2 \leqq c'\|\mathbf{f}_2\| n^{(1-m-\alpha)/3},\end{aligned}$$

provided $m + \alpha > 1$. Further,

$$(3.19b)\qquad \begin{aligned} &\|(I + C_n^{(3)} + B_n^{(3)} + B_n^{(4)})\tilde{\boldsymbol\sigma} - \mathbf{f}\|_\infty \leqq c'\|\mathbf{f}\|_\infty n^{5/6-(m+\alpha)/3},\\ &\|\tilde{\boldsymbol\sigma} - \boldsymbol\sigma^*\|_\infty \leqq c'\|\mathbf{f}\|_\infty n^{5/6-(m+\alpha)/3}.\end{aligned}$$

Proof. We solve (3.18) by solving

$$(3.20)\qquad (I + D_n^{(3)} + H_n^{(5)})\hat{\boldsymbol\sigma} = W_n\mathbf{f}, \qquad \tilde{\boldsymbol\sigma} = W_n^H\hat{\boldsymbol\sigma}.$$

The elements of $H_n^{(5)}$ are determined in $O(l^2 \log l) = O(n^{2/3} \log n)$ arithmetic operations, and the LU-decomposition is determined in $O(l^3) = O(n)$ arithmetic operations. The asymptotic operation count of $O(n \log n)$ stems from the computation of $W_n\mathbf{f}$ and $W_n^H\hat{\boldsymbol\sigma}$. We turn to (3.19a). By (3.14) and (3.16a) there is a constant c'', independent of l and of n sufficiently large, such that

$$(3.21)\qquad \begin{aligned} \|(I + C_n^{(3)} + B_n^{(3)} + B_n^{(4)})\tilde{\boldsymbol\sigma} - \mathbf{f}\|_2 &\leqq \|B_n^{(4)}\|_2\|\tilde{\boldsymbol\sigma}\|_2\\ &\leqq \|B_n^{(4)}\|_2\|(I + C_n^{(3)} + B_n^{(3)})^{-1}\|_2\|\mathbf{f}\|_2\\ &\leqq c''\|\mathbf{f}\|_2 l^{-m-\alpha+1}.\end{aligned}$$

Let $l_1 > \max\{l_0, c''^{1/(m+\alpha-1)}\}$. Then for $n \geqq l \geqq l_1$ $(I + C_n^{(3)} + B_n^{(3)})^{-1}$ exists and

$$\|(I + C_n^{(3)} + B_n^{(3)})^{-1}B_n^{(4)}\|_2 \leqq c'' l^{-m-\alpha+1} < 1.$$

Therefore

$$(3.22)\qquad \begin{aligned} \|\tilde\sigma - \sigma^*\|_2 &= \|(I + C_n^{(3)} + B_n^{(3)})^{-1}\mathbf{f} - (I + C_n^{(3)} + B_n^{(3)} + B_n^{(4)})^{-1}\mathbf{f}\|_2\\ &= \Big\|\sum_{k=1}^{\infty} (-1)^k((I + C_n^{(3)} + B_n^{(3)})^{-1}B_n^{(4)})^k(I + C_n^{(3)} + B_n^{(3)})^{-1}\mathbf{f}\Big\|_2\\ &\leqq \sum_{k=1}^{\infty} (c'' l^{-m-\alpha+1})^k c\|\mathbf{f}\|_2\\ &\leqq c\|\mathbf{f}\|_2 l^{-m-\alpha+1}\end{aligned}$$

where the constant c is independent of $l \geqq l_1$ and $n \geqq n_1$. c is the bound in (3.16). Formula (3.19a) now follows from (3.21), (3.22), and $l^{-1} \leqq d_1^{-1} n^{-1/3}$. Condition $m + \alpha > 1$ was sufficient when deriving (3.14) and (3.16). Formula (3.19b) is obtained from (3.19a) by noting that for any vector $\mathbf{v} \in \mathbb{C}^n$,

$$(3.23)\qquad \|\mathbf{v}\|_\infty \leqq \|\mathbf{v}\|_2 \leqq n^{1/2}\|\mathbf{v}\|_\infty. \qquad \square$$

Let $\tilde f(s) \in C^{m'+\alpha'}[0, 2\pi]$ with $0 \leqq m' + \alpha' < m + \alpha$. Then $\tilde\sigma(s)$, the solution of (3.4), satisfies $\tilde\sigma(s) \in C^{m'+\alpha'}[0, 2\pi]$ but, generally, $\tilde\sigma(s) \notin C^\mu[0, 2\pi]$ for $\mu > m' + \alpha'$. Let $\hat\sigma(s)$ be the trigonometric polynomial of degree at most n of best approximation of $\tilde\sigma(s)$

with respect to the uniform norm on $[0, 2\pi]$. Then (see [Me, Satz 41]),

$$\max_{0 \leqq s \leqq 2\pi} |\tilde{\sigma}(s) - \hat{\sigma}(s)| = O(n^{-m'-\alpha'}),$$

and the exponent can generally not be improved [Me, Satz 48]. We therefore say that $\tilde{\boldsymbol{\sigma}}$ *converges optimally* to $\boldsymbol{\sigma}^*$ if $\|\tilde{\boldsymbol{\sigma}} - \boldsymbol{\sigma}^*\|_\infty = O(n^{-m'-\alpha'})$. By (3.19b) a sufficient condition for optimal convergence is

$$m + \alpha \geqq 3(m' + \alpha') + \tfrac{5}{2}.$$

Hence, optimal convergence of $\tilde{\boldsymbol{\sigma}}$ is achieved if the kernel is sufficiently much smoother than the right-hand side function $\tilde{f}(s)$.

If n is not large enough so that the matrix $B_n^{(4)}$ can be dropped from (3.15) without yielding poor accuracy, we solve (3.15) by the block Jacobi method

$$\text{(3.24)}\quad \begin{aligned} &W_n^H(I + D_n^{(3)} + H_n^{(5)}) W_n \boldsymbol{\sigma}^{(k+1)} = -B_n^{(4)} \boldsymbol{\sigma}^{(k)} + \mathbf{f}, \qquad k = 0, 1, \cdots, \\ &W_n^H(I + D_n^{(3)} + H_n^{(5)}) W_n \boldsymbol{\sigma}^{(0)} = \mathbf{f}. \end{aligned}$$

Each iteration requires $O(n^2)$ arithmetic operations including two fast Fourier transforms. Let $l = l(n) \leqq d_2 n^{2/3}$ for some constant $d_2 > 0$. Then $I + D^{(3)} + H_n^{(5)}$ and its LU-decomposition can be computed in $O(n^2)$ arithmetic operations. By not forming $B_n^{(4)}$ explicitly, but by instead using the right-hand side of

$$B_n^{(4)} = \frac{2}{n} \operatorname{Re}(B_n) - W_n^H(I + D_n^{(3)} + H_n^{(5)}) W_n,$$

we can determine the linear system (3.24) in $O(n^2)$ arithmetic operations.

COROLLARY 3.3. *The rate of convergence of the iterations* (3.24) *increases with n provided that* $m + \alpha > 1$.

Proof. By (3.14) and (3.16) there is a constant c'' such that for $l \geqq d_1 n^{1/3}$ and for n sufficiently large

$$\text{(3.25)}\qquad \|[I + C_n^{(3)} + B_n^{(3)}]^{-1} B_n^{(4)}\|_2 \leqq c'' d_1^{1-m-\alpha} n^{(1-m-\alpha)/3}.$$

The condition $m + \alpha > 1$ stems from the derivation of (3.14) and (3.16). □

Remark. Let $c > 0$ and $\gamma > 0$ be arbitrary constants, and let $\boldsymbol{\sigma}_n^*$ be as in Theorem 3.2. Assume for the moment that the $\boldsymbol{\sigma}_n^{(k)}$ can be computed without roundoff errors. Then it follows from (3.25) that there is an integer $p \geqq 0$, depending on l, c, and γ, but independent of n, such that

$$\|\boldsymbol{\sigma}_n^{(k)} - \boldsymbol{\sigma}_n^*\|_2 \leqq c n^{-\gamma}, \qquad k \geqq p.$$

Hence, if the discretization error is of the form $O(n^{-\gamma})$, then the error in $\boldsymbol{\sigma}_n^{(k)}$ is of the same order of magnitude as the discretization error after a number of iterations, which is independent of n. Similar results hold for the multigrid method [Ha], [HS]. □

Numerical experiments with a preconditioned conjugate gradient method show even faster convergence than iterations (3.24), despite the fact that the matrix $I + C_n^{(3)} + B_n^{(3)} + B_n^{(4)}$ generally is nonsymmetric. The conjugate gradient method has been implemented as described in [GVL, (10.3-3)] with preconditioning matrix $I + C_n^{(3)} + B_n^{(3)}$. Computed examples are presented in § 4.

3.2. Exterior Dirichlet problems. Let Ω in (3.1) be the region exterior to Γ. Now the positive direction of Γ is clockwise. Equation (1.8) with $f(z) \equiv 0$ has solution $\sigma \equiv 1$, but no other linearly independent solution. Let $z'(t)$ be the same as in (3.2). It is easy to see that the following system of equations for the exterior Dirichlet problem has a unique solution $\{q^*, \sigma^*\} \in \mathbb{R} \times L^2(\Gamma)$:

$$(3.26)\qquad \begin{aligned} &q + \tilde{\sigma}(s) + \frac{1}{\pi} \operatorname{Im}\left(\int_0^{2\pi} \frac{z'(t)}{z(s)-z(t)} \tilde{\sigma}(t)\,dt\right) = \tilde{f}(s), \qquad 0 \leqq s \leqq 2\pi, \\ &\int_0^{2\pi} \tilde{\sigma}(t)\,dt = 0. \end{aligned}$$

Discretize (3.26) by the trapezoidal rule in the same way as (3.4) was discretized. Let $\mathbf{e} := (1, 1, \cdots, 1)^T$. Similarly as (3.15), we obtain

$$(3.27)\qquad \begin{aligned} &q\mathbf{e} + (I - C_n^{(3)} - B_n^{(3)})\boldsymbol{\sigma} = B_n^{(4)}\boldsymbol{\sigma} + \mathbf{f}, \\ &\mathbf{e}^T\boldsymbol{\sigma} = 0. \end{aligned}$$

Let $\hat{\boldsymbol{\sigma}} = (\hat{\sigma}_0, \hat{\sigma}_1, \cdots, \hat{\sigma}_{n-1})^T$ be the scaled Fourier coefficients $\hat{\boldsymbol{\sigma}} := W_n\boldsymbol{\sigma}$. Regard $\hat{\boldsymbol{\sigma}}$ as the unknown to be determined. Let $\mathbf{e}_1 := (1, 0, \cdots, 0)^T$. From (3.27), we obtain

$$(3.28a)\qquad q\mathbf{e}_1\sqrt{n} + (I - D_n^{(3)} - H_n^{(5)})\hat{\boldsymbol{\sigma}} = W_n B_n^{(4)} W_n^H \hat{\boldsymbol{\sigma}} + W_n\mathbf{f},$$

$$(3.28b)\qquad \hat{\sigma}_0 = 0.$$

Introduce the matrix $P = [p_{jk}] \in \mathbb{R}^{n\times(n-1)}$ with

$$p_{jk} := \begin{cases} 0, & j \neq k+1, \quad 0 \leqq j < n, \quad 0 \leqq k < n-1, \\ 1, & j = k+1, \quad 0 \leqq k < n-1. \end{cases}$$

From (3.28), $P\mathbf{e}_1 = \mathbf{0}$, and $P^T D^{(3)} P = 0$, it follows that

$$(3.29)\qquad (I - P^T H^{(5)} P)\begin{bmatrix} \hat{\sigma}_1 \\ \vdots \\ \hat{\sigma}_{n-1} \end{bmatrix} = P^T W_n B_n^{(4)} W_n^H P \begin{bmatrix} \hat{\sigma}_1 \\ \vdots \\ \hat{\sigma}_{n-1} \end{bmatrix} + P^T W_n \mathbf{f}.$$

From (3.29) we compute $(\hat{\sigma}_1, \cdots, \hat{\sigma}_{n-1})^T$ in the same way as (3.15) was solved. Knowing $\hat{\boldsymbol{\sigma}}$, we can determine q from the first row of (3.28a).

Also for Dirichlet problems on multiply connected regions the matrix has a structure that enables the use of the solution methods described. This structure has been used previously in [Re] to obtain a different iterative scheme. We omit the details.

3.3. Cauchy singular integral equations. We consider the integral equations that may be the most closely related to the matrix problem of § 2. Let Γ be smooth and regard the integral equation for complex-valued σ:

$$\frac{1}{2}\sigma(z) + \frac{1}{2\pi i} PV \int_\Gamma \frac{\sigma(\zeta)}{z-\zeta}\,d\zeta = f(z), \qquad z \in \Gamma.$$

Introducing a parametric representation and using the same notation as in (3.3), we obtain

$$(3.30)\qquad \tilde{\sigma}(s) + \frac{1}{\pi} PV \int_0^{2\pi} \frac{e^{it}}{e^{is}-e^{it}} \tilde{\sigma}(t)\,dt - \frac{1}{\pi}\int_0^{2\pi} b(s,t)\tilde{\sigma}(t)\,dt = \tilde{f}(s), \qquad 0 \leqq s \leqq 2\pi.$$

We discretize (3.30) by a collocation method with collocation points $s_j = 2\pi j/n, 0 \leqq j < n$. Let $t_j := s_j$. The integral with kernel $b(s, t)$ we discretize by the trapezoidal rule with nodes t_j, $0 \leqq j < n$. The principal value integral we integrate by a rule obtained from [He, (14.2-35o)], which for n even and $0 \leqq j < n$ reads

$$\begin{aligned}\frac{1}{\pi} PV \int_0^{2\pi} \frac{e^{it}}{e^{it_j} - e^{it}} \tilde{\sigma}(t)\, dt \approx \rho_j &:= \frac{2}{n} \sum_{\substack{k=0 \\ k \neq j}}^{n-1} \frac{e^{it_k}}{e^{it_j} - e^{it_k}} \sigma_k \\ &\quad - \frac{(-1)^j}{n} \sum_{\substack{k=0 \\ k \neq j}}^{n-1} (-1)^k \frac{e^{it_j} + e^{it_k}}{e^{it_j} - e^{it_k}} \sigma_k - \frac{\sigma_j}{n},\end{aligned} \tag{3.31}$$

where $\sigma_k := \tilde{\sigma}(t_k)$. This rule is derived by interpolating $\tilde{\sigma}(t)$ by a trigonometric polynomial that is integrated exactly [He, Chap. 14.2.V]. Let $\boldsymbol{\rho} := (\rho_0, \rho_1, \cdots, \rho_{n-1})^T$ and let $C_n^{(4)}$ be the circulant matrix implicitly defined by (3.31), such that $\boldsymbol{\rho} = C_n^{(4)} \boldsymbol{\sigma}$. Then $D_n^{(4)} := W_n C_n^{(4)} W_n^H$ is a diagonal matrix whose elements can be determined explicitly using the formulas of the Appendix. The discretization of (3.30) yields

$$\left(I + C_n^{(4)} + \frac{2}{n} B_n \right) \boldsymbol{\sigma} = \mathbf{f}$$

or equivalently with $\hat{\boldsymbol{\sigma}} = W_n \boldsymbol{\sigma}$,

$$\left(I + D^{(4)} + \frac{2}{n} W_n B_n W_n^H \right) \hat{\boldsymbol{\sigma}} = W_n \mathbf{f}.$$

Splitting $(2/n) W_n B_n W_n^H$ as in § 3.1, we obtain solution methods similar to those already discussed.

4. Numerical examples. The computations were carried out on an Alliant FX/8 computer in double precision arithmetic, i.e., with 15 significant digits. We first illustrate the approximation of matrices $A_n \in \mathbb{C}^{n \times n}$ by matrices $A_n^{(2)} \in \mathbb{C}^{n \times n}$ of rank $l = l(n) \ll n$. The notation of § 2 is used, and hence, we must bound $G_n^{(2)} - G_n^{(5)}$. From (2.18) we then can compute a bound for $A_n - A_n^{(2)}$.

Example 4.1. Let Γ be the ellipse with parametric representation $z(t) = 2 \cos(t) + i \sin(t)$, $0 \leqq t \leqq 2\pi$. Choose $l = 2n^{1/2}$. The difference $G_n^{(2)} - G_n^{(5)}$ is shown in Table 4.1. Table 4.2 illustrates the use of a larger l. We choose $l = 4n^{1/2}$. □

TABLE 4.1

n	l	$\|G_n^{(2)} - G_n^{(5)}\|_F$	$\|G_n^{(2)} - G_n^{(5)}\|_\infty$
36	12	7.4 (−2)	9.9 (−2)
64	16	1.5 (−2)	2.0 (−2)
100	20	2.5 (−3)	3.4 (−3)

TABLE 4.2

n	l	$\|G_n^{(2)} - G_n^{(5)}\|_F$	$\|G_n^{(2)} - G_n^{(5)}\|_\infty$
36	24	1.0 (−4)	1.4 (−4)
64	32	2.2 (−6)	3.0 (−6)
100	40	4.3 (−8)	5.7 (−8)

Example 4.2. Let Γ be the curve shown in Fig. 1 with parametric representation $z(t)=x(t)+iy(t)$, $0\leqq t\leqq 2\pi$, where

$$x(t):=1.225(0.5\cos(t)+\cos(2t)-1),$$

$$y(t):=1.75[0.1225(5\sin(t-0.2)+2\sin(2t)-\sin(4t))+0.4\sin(t)-0.185].$$

For this curve a fairly large constant c in $l:=cn^{1/2}$ is required in order to make $G_n^{(2)}-G_n^{(5)}$ small for moderate n. We choose $c=8$, and show $G_n^{(2)}-G_n^{(5)}$ so obtained in Table 4.3. □

Example 4.3. Let Γ be the race-track-shaped curve that is the boundary of a region obtained by placing a 0.2×2 rectangle between two unit disk halves (see Fig. 2). The parametric representation $z(t)$, $0\leqq t\leqq 2\pi$, of Γ is defined by letting t be proportional to the arclength from $z(0)$ when Γ is traversed in the positive direction. $z(0)$ is chosen as the midpoint of one of the circular arcs. $z(t)$ does not satisfy the smoothness requirements of § 2; $z''(t)$ has jump discontinuities. We choose $l:=2n^{1/2}$, which is the same choice as for Table 4.1, and obtain Table 4.4. The difference $G_n^{(2)}-G_n^{(5)}$ is seen to be large, and appears to grow with n. □

The remaining examples illustrate application of the matrix splitting to the solution of the integral equation of § 3.1. We use the notation of § 3 and also introduce for any

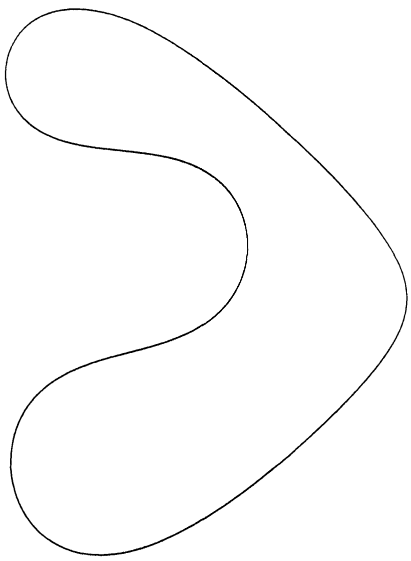


FIG. 1

TABLE 4.3

n	l	$\lVert G_n^{(2)}-G_n^{(5)}\rVert_F$	$\lVert G_n^{(2)}-G_n^{(5)}\rVert_\infty$
100	80	4.6 (−2)	5.2 (−2)
144	96	1.5 (−2)	1.7 (−2)
196	112	4.9 (−3)	5.2 (−3)
256	128	1.5 (−3)	1.6 (−3)
314	144	4.5 (−4)	4.6 (−4)

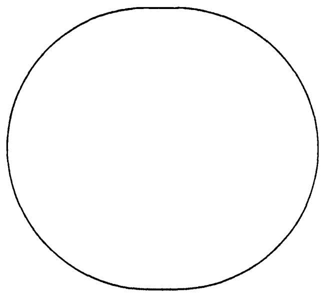


FIG. 2

TABLE 4.4

n	l	$\|G_n^{(2)} - G_n^{(5)}\|_F$	$\|G_n^{(2)} - G_n^{(5)}\|_\infty$
36	12	1.6	2.2
64	16	2.3	3.5
100	20	2.4	3.7
144	24	2.7	4.7

$\boldsymbol{\sigma} = (\sigma_0, \sigma_1, \cdots, \sigma_{n-1})^T \in \mathbb{R}^n$ and $0 \leqq s \leqq 2\pi$,

$$(K_n\boldsymbol{\sigma})(s) := \frac{1}{n}\sum_{j=0}^{n-1} \sigma_j + \frac{2}{n}\sum_{j=0}^{n-1} \operatorname{Re}\left(\frac{e^{it_j}}{e^{is} - e^{it_j}} + i\frac{z'(t_j)}{z(s) - z(t_j)}\right)\sigma_j, \tag{4.1}$$

$$\boldsymbol{\sigma}(s) := \left(\sum_{j=0}^{n-1} (-1)^j \sigma_j \cot\left(\frac{s - t_j}{2}\right)\right) \Big/ \left(\sum_{j=0}^{n-1} (-1)^j \cot\left(\frac{s - t_j}{2}\right)\right), \tag{4.2}$$

where $t_j := 2\pi j/n$. Formula (4.2) is a barycentric formula for trigonometric polynomial interpolation [He, Chap. 13.6]. $\boldsymbol{\sigma}(s)$ is a trigonometric polynomial of degree $\leqq n/2$ such that $\boldsymbol{\sigma}(t_j) = \sigma_j$, $0 \leqq j < n$.

Let $\boldsymbol{\sigma}^*$ solve (3.5) and let $\tilde{\boldsymbol{\sigma}}$ satisfy (3.18). Then

$$\mathbf{e}^*(s) := \boldsymbol{\sigma}^*(s) + (K_n\boldsymbol{\sigma}^*)(s) - 2f(s) \tag{4.3a}$$

is the residual due to discretization errors and

$$\tilde{\mathbf{e}}(s) := \tilde{\boldsymbol{\sigma}}(s) + (K_n\tilde{\boldsymbol{\sigma}})(s) - 2f(s) \tag{4.3b}$$

is the residual due to discretization and the approximation of $(2/n)\operatorname{Re}(B_n)$ by $B_n^{(3)}$. We would like $\tilde{\mathbf{e}}(s) \approx \mathbf{e}^*(s)$. The size of $\mathbf{e}^*(s)$ is measured by the discrete uniform norm

$$\|\mathbf{e}^*\|_d := \max_{0 \leqq j < 4n} |\mathbf{e}^*(\pi j/(2n))|, \qquad \mathbf{e}^*(s) \in \mathbb{R}^n,$$

and the same norm is used for $\tilde{\mathbf{e}}_n(s)$. In Examples 4.4–4.6 we let

$$f(t) := |\sin(2t)|^{3/2},$$

a fairly smooth but nonanalytic function.

We note that since all matrices and vectors are real-valued, we can use real FFTs. The unitary Fourier matrix (1.2) can be replaced by the orthonormal Fourier matrix

$W_n = [w_{jk}]$ that for n even is defined by

$$w_{0,k} := n^{-1/2}, \quad 0 \leqq k < n,$$

$$\left.\begin{aligned} w_{2j-1,k} &:= \left(\frac{2}{n}\right)^{1/2} \sin\left(\frac{2\pi jk}{n}\right), \\ w_{2j,k} &:= \left(\frac{2}{n}\right)^{1/2} \cos\left(\frac{2\pi jk}{n}\right), \end{aligned}\right\} \quad 1 \leqq j < \frac{n}{2}, \quad 0 \leqq k < n.$$

For n odd, an analogous orthonormal matrix W_n can be used.

Example 4.4. Let Γ be the ellipse with parametric representation $z(t) := 2\cos(t) + i\sin(t)$. Let l be the even integer closest to $2n^{1/3}$. We obtain Table 4.5, which shows that $(2/n)\operatorname{Re}(B_n)$ can be replaced by $B_n^{(3)}$ with almost no loss of accuracy. □

Example 4.5. Let Γ and $z(t)$ be the same as in Example 4.2, and choose l as the even integer closest to $8n^{1/3}$. We obtain Table 4.6, which shows that $(2/n)\operatorname{Re}(B_n)$ can be replaced by $B_n^{(3)}$ with almost no loss of accuracy. □

Example 4.6. Let Γ be the race-track-shaped boundary of a region obtained by placing a 2×2 square between two unit disk halves (see Fig. 3). $z(t)$ is obtained similarly as in Example 4.3 and l is chosen as in Example 4.5. We obtain Table 4.7. While the theory does not cover this example, we nevertheless obtain an acceptable approximation $B_n^{(3)}$ of $(2/n)\operatorname{Re}(B_n)$, i.e., replacing $(2/n)\operatorname{Re}(B_n)$ by $B_n^{(3)}$ yields $\mathbf{e}^*(s) \approx \tilde{\mathbf{e}}(s)$. □

In all the above integral equation examples we have been able to replace $(2/n)\operatorname{Re}(B_n)$ by $B_n^{(3)}$ and obtain $\tilde{\mathbf{e}}(s) \approx \mathbf{e}^*(s)$. In the next example this is not the case,

TABLE 4.5

n	$\|\mathbf{e}^*\|_d$	l	$\|\sigma^* - \tilde{\sigma}\|_\infty$	$\|\sigma^* - \tilde{\sigma}\|_2$	$\|\tilde{\mathbf{e}}\|_d$
64	1.2 (−2)	8	1.1 (−2)	6.4 (−2)	1.6 (−2)
128	4.1 (−3)	10	1.3 (−3)	1.0 (−2)	3.7 (−3)
256	1.5 (−3)	12	1.6 (−4)	1.6 (−3)	1.3 (−3)

TABLE 4.6

n	$\|\mathbf{e}^*\|_d$	l	$\|\sigma^* - \tilde{\sigma}\|_\infty$	$\|\sigma^* - \tilde{\sigma}\|_2$	$\|\tilde{\mathbf{e}}\|_d$
64	1.2 (−2)	32	1.3 (−2)	2.7 (−2)	1.6 (−2)
128	4.1 (−3)	40	3.4 (−3)	8.2 (−3)	4.4 (−3)
256	1.5 (−3)	50	4.4 (−4)	1.7 (−3)	1.6 (−3)

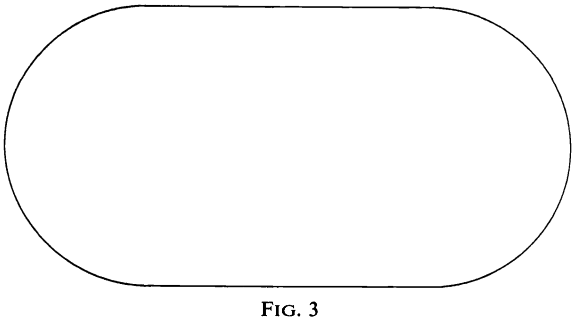


FIG. 3

TABLE 4.7

n	$\|\mathbf{e}^*\|_d$	l	$\|\sigma^* - \tilde{\sigma}\|_\infty$	$\|\sigma^* - \tilde{\sigma}\|_2$	$\|\tilde{\mathbf{e}}\|_d$
64	1.2 (−2)	32	1.7 (−2)	5.3 (−2)	1.9 (−2)
128	4.1 (−3)	40	1.6 (−2)	5.8 (−2)	1.7 (−2)
256	1.5 (−3)	50	4.9 (−3)	2.8 (−2)	5.3 (−3)

and we use $I + C_n^{(3)} + B_n^{(3)}$ as a preconditioning matrix in block Jacobi and conjugate gradient iterations.

Example 4.7. Let Γ and $z(t)$ be the same as in Example 4.6, but let $f(t)$ now be the smooth function

$$f(t) := |\sin(2t)|^{9/2}.$$

Let $\sigma^{(j)}$ be defined by the block Jacobi iterations (3.25) or by the preconditioned conjugate gradient method described in [GVL, (10.3-3)] with preconditioner $I + C_n^{(3)} + B_n^{(3)}$, which is LU factorized. $\sigma^{(0)}$ for the conjugate gradient (cg) method is given by (3.25). Despite the fact that the matrix $I + C_n^{(3)} + (2/n)\,\mathrm{Re}\,(B_n)$ as well as the preconditioner are nonsymmetric, the *cg* iterations converge slightly faster than the Jacobi iterations. This observation has been made in many computed examples. If nonsymmetry would cause slow or no convergence of the *cg* iterations, the method described in [EES] could be used. Let

$$\mathbf{e}^{(j)}(s) := \sigma^{(j)}(s) + (K_n\sigma^{(j)})(s) - 2f(s).$$

Choose $n = 64$, $l := 8n^{1/3} = 32$. Table 4.8 shows the convergence of the Jacobi iterations (3.25). For comparison, $\|\mathbf{e}^*\|_d = 7.0(-5)$.

After two iterations $\|\mathbf{e}^{(j)}\|_d \approx \|\mathbf{e}^*\|_d$. Letting $n := 128$, $l := 40 \approx 8n^{1/3}$, we obtain Table 4.9.

For $n = 128$, we have $\|\mathbf{e}^*\|_d = 1.6(-6) \approx \|\mathbf{e}^{(2)}\|_d$. Table 4.10 and 4.11 show *cg* iterations and correspond to Tables 4.8 and 4.9, respectively. □

TABLE 4.8
$n = 64$, $l = 32$, *Jacobi iterations.*

j	$\|\sigma^* - \sigma^{(j)}\|_x$	$\|\sigma^* - \sigma^{(j)}\|_2$	$\|\mathbf{e}^{(j)}\|_d$
0	2.0 (−2)	3.4 (−2)	1.7 (−2)
1	5.4 (−4)	1.2 (−3)	4.1 (−4)
2	1.4 (−5)	2.2 (−5)	7.4 (−5)

TABLE 4.9
$n = 128$, $l = 40$, *Jacobi iterations.*

j	$\|\sigma^* - \sigma^{(j)}\|_\infty$	$\|\sigma^* - \sigma^{(j)}\|_2$	$\|\mathbf{e}^{(j)}\|_d$
0	3.5 (−3)	9.5 (−3)	3.0 (−3)
1	5.8 (−5)	1.4 (−4)	4.2 (−5)
2	5.5 (−7)	1.4 (−6)	1.7 (−6)

TABLE 4.10
$n=64$, $l=32$, cg *iterations.*

j	$\|\sigma^*-\sigma^{(j)}\|_\infty$	$\|\sigma^*-\sigma^{(j)}\|_2$	$\|e^{(j)}\|_d$
0	2.0 (−2)	3.4 (−2)	1.7 (−2)
1	4.8 (−4)	1.1 (−3)	3.7 (−4)
2	4.3 (−6)	8.6 (−6)	7.1 (−5)

TABLE 4.11
$n=128$, $l=40$, cg *iterations.*

j	$\|\sigma^*-\sigma^{(j)}\|_\infty$	$\|\sigma^*-\sigma^{(j)}\|_2$	$\|e^{(j)}\|_d$
0	3.5 (−3)	9.5 (−3)	3.0 (−3)
1	3.9 (−5)	9.6 (−5)	2.8 (−5)
2	1.7 (−7)	4.4 (−7)	1.6 (−6)

Appendix. Proof of (1.4).

By [Da, 3.2.2] the elements of $D_n^{(2)}=W_nA_nD_nW_n^H$ can be written as follows:

$$D_n^{(2)}=\operatorname{diag}(p(1),\ p(z_1^{n-1}),p(z_1),p(z_1^{n-2}),\ p(z_1^2),\cdots,p(z_1^{n/2-1}),p(z_1^{n/2})),$$

where $p(z)=\sum_{k=0}^{n-1}a_{0k}z_kz^k$, and $z_k=e^{2\pi ik/n}$. (Note that matrix W defined by (1.2) is obtained by reordering the rows of the Fourier matrix used in [Da].) The diagonal elements can be computed explicitly. For $0\leqq j<n$,

$$p(z_1^j)=\sum_{k=1}^{n-1}\frac{z_k}{1-z_k}z_1^{jk}=\sum_{k=1}^{n-1}\frac{z_k}{1-z_k}z_k^j=\sum_{k=1}^{n-1}\frac{z_k^{j+1}-1}{1-z_k}+\sum_{k=1}^{n-1}\frac{1}{1-z_k}.$$

Now

$$\sum_{k=1}^{n-1}\frac{z_k^{j+1}-1}{1-z_k}=-\sum_{k=1}^{n-1}\frac{z_k^{j+1}-1}{z_k-1}=-\sum_{k=1}^{n-1}\sum_{l=0}^{j}z_k^l$$

$$=-\sum_{k=0}^{n-1}\left(\sum_{l=0}^{j}z_k^l\right)+j+1=-\sum_{l=0}^{j}\left(\sum_{k=0}^{n-1}z_k^l\right)+j+1=-n+j+1$$

where the last equality follows by the orthogonality of z^l to 1 for $1\leqq l<n$. From

$$\sum_{k=1}^{n-1}\frac{1}{1-z_k}=\sum_{k=1}^{n/2-1}\left(\frac{1}{1-z_k}+\frac{1}{1-\bar z_k}\right)+\frac{1}{2}=\sum_{k=1}^{n/2-1}\frac{1-\bar z_k+1-z_k}{1-z_k-\bar z_k+1}+\frac{1}{2}$$

$$=\frac{n}{2}-1+\frac{1}{2}=\frac{n-1}{2},$$

we obtain $p(z_1^j)=j-(n-1)/2$, $0\leqq j<n$, and (1.4) follows. □

Acknowledgments. Work on this paper was begun while I visited the Computer Science Department at Stanford University. I thank Gene Golub for the kind hospitality and for introducing me to the matrix problem discussed.

The paper was finished during a semester at the University of South Florida. I thank Ed Saff for making my stay there possible.

Finally, I would like to thank the referees for carefully reading the manuscript, and for suggesting improvements of the presentation.

REFERENCES

[At] K. E. Atkinson, *A Survey of Numerical Methods for the Solution of Fredholm Integral Equations of the Second Kind*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1976.

[Be] J.-P. Berrut, *Integralgleichungen und Fourier-Methoden zur numerischen konformen Abbildung*, Ph.D. thesis, ETH, Zurich, Switzerland, 1985.

[Da] P. J. Davis, *Circulant Matrices*, John Wiley, New York, 1979.

[DF] L. M. Delves and T. L. Freeman, *Analysis of Global Expansion Methods: Weakly Asymptotically Diagonal Systems*, Academic Press, London, 1981.

[EES] S. C. Eisenstat, H. C. Elman, and M. H. Schultz, *Variational iterative methods for nonsymmetric systems of linear equations*, SIAM J. Numer. Anal., 20 (1983), pp. 345-357.

[Ge] A. Gerasoulis, *A fast algorithm for the multiplication of generalized Hilbert matrices with vectors*, Math. Comput., 50 (1988), pp. 179-188.

[GGS] A. Gerasoulis, M. D. Grigoriadis, and Liping Sun, *A fast algorithm for Trummer's problem*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s135-s138.

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

[GT] G. H. Golub and M. R. Trummer, *Communication on NA-net*, 1985.

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

[Ha] W. Hackbusch, *Multigrid methods of the second kind*, in Multigrid Methods for Integral and Differential Equations, D. J. Paddon and H. Holstein, eds., Clarendon Press, Oxford, 1985.

[HS] P. W. Hemker and H. Schippers, *Multiple grid methods for the solution of Fredholm integral equations of the second kind*, Math. Comput., 36 (1981), pp. 215-232.

[He] P. Henrici, *Applied and Computational Complex Analysis*, Vol. 3, John Wiley, New York, 1986.

[Ka] Y. Katznelson, *An Introduction to Harmonic Analysis*, John Wiley, New York, 1968.

[LSW] U. Lamp, K.-T. Schleicher, and W. L. Wendland, *The fast Fourier transform and the numerical solution of one-dimensional integral equations*, Numer. Math., 47 (1985), pp. 15-38.

[Me] G. Meindardus, *Approximation von Funktionen und ihre numerische Behandlung*, Springer-Verlag, Berlin, New York, 1964.

[Re] L. Reichel, *A fast method for solving certain integral equations of the first kind with application to conformal mapping*, J. Comput. Appl. Math., 14 (1986), pp. 125-142.

[Ro] V. Rokhlin, *Rapid solution of integral equations of classical potential theory*, J. Comput. Phys., 60 (1985), pp. 187-207.

[Sch] H. Schippers, *Multigrid methods for boundary integral equations*, Numer. Math., 46 (1985), pp. 315-363.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 2, pp. 281-292, March 1990

A BRANCH AND BOUND ALGORITHM FOR THE BILEVEL PROGRAMMING PROBLEM*

JONATHAN F. BARD† AND JAMES T. MOORE‡

Abstract. The bilevel programming problem is a static Stackelberg game in which two players try to maximize their individual objective functions. Play is sequential and uncooperative in nature. This paper presents an algorithm for solving the linear/quadratic case. In order to make the problem more manageable, it is reformulated as a standard mathematical program by exploiting the follower's Kuhn-Tucker conditions. A branch and bound scheme suggested by Fortuny-Amat and McCarl is used to enforce the underlying complementary slackness conditions. An example is presented to illustrate the computations, and results are reported for a wide range of problems containing up to 60 leader variables, 40 follower variables, and 40 constraints. The main contributions of the paper are in the step-by-step details of the implementation, and in the scope of the testing.

Key words. bilevel programming, Stackelberg games, branch and bound, complementarity sequential game, linear programming

AMS(MOS) subject classifications. 65-03, 90D05, 90C05

1. Introduction. Hierarchical decision problems involving conflict among the different subunits can often be modeled as a multilevel game. Examples include government regulation, management of a decentralized firm, and the standard max-min problem. In each of these situations, the leader attempts to maximize his objective function by selecting a strategy that anticipates the reactions of the followers. In so doing, if he is limited to influencing rather than controlling subunit outcomes, a Stackelberg game results (see Basar and Selbuz (1979), or Simaan and Cruz (1973)). The bilevel programming problem (BLPP) is a static version of this game where the leader has control over the decision variables $x \in X \subseteq R^{n_1}$, while the follower separately controls the decision variables $y \in Y \subseteq R^{n_2}$ (Aiyoshi and Shimizu (1981), Bard and Falk (1982), Bialas and Karwan (1984)).

In our formulation, it will be assumed that the leader goes first and chooses an x to maximize his objective function $F(x, y)$. The follower then reacts by selecting a y to maximize his individual objective function $f(x, y)$ without regard to the external consequences of his actions. Here, $F: X \times Y \to R^1$ and $f: X \times Y \to R^1$. The problem addressed in this paper is the "linear/quadratic" case given by

$$\max_x F(x, y) = c^1x + c^2y, \tag{1a}$$

$$\text{subject to} \quad x \in X = \{x: Dx \geqq d\}, \tag{1b}$$

$$\max_y f(x, y) = c^3y + x^TQ_1y + \tfrac{1}{2}y^TQ_2y, \tag{1c}$$

$$\text{subject to} \quad g(x, y) = Ax + By \geqq b, \tag{1d}$$

$$y \in Y = \{y: Ey \geqq e\} \tag{1e}$$

* Received by the editors April 13, 1987; accepted for publication (in revised form) March 13, 1989.

† Operations Research Group, Department of Mechanical Engineering, University of Texas, Austin, Texas 78712.

‡ Major, U.S. Air Force, Strategic Air Command Headquarters, Offutt Air Force Base, Omaha, Nebraska 68113.

where A is $m_1 \times n_1$, B is $m_1 \times n_2$, D is $m_2 \times n_1$, E is $m_3 \times n_2$, Q_1 is $n_1 \times n_2$, Q_2 is $n_2 \times n_2$ symmetric, negative semidefinite, and c^1, c^2, c^3, b, d, and e are vectors of conformal dimension. Note that it is always possible to drop components separable in x from the follower's objective function without altering the results. Hence, (1c) does not contain linear and quadratic terms in x.

A significant amount of effort has been devoted to solving (1a)-(1e), which Jeroslow (1985) has shown to be NP-*hard*. A survey can be found in Bialas and Karwan (1984) where they outline both their "high point" algorithm and complementary pivot approach. Also see Candler and Townsley (1982) and Fortuny-Amat and McCarl (1981); for a generalization of (1a)-(1e) to many players see Bard (1983a) and Gardner and Cruz (1978).

The intent of this paper is to describe the implementation and testing of a branch and bound scheme for solving (1a)-(1e) initially suggested by Fortuny-Amat and McCarl. The approach is similar to that developed by Bard (1988) for dealing with the case where $F(x, y)$ is strictly concave, but is more robust as discussed in § 3. In the next section, terminology and assumptions are presented. This is followed in § 3 by the development of the algorithm and an example to highlight its operations; § 4 contains our computational experience and some insights gained from testing. We conclude in § 5 with a discussion of the results.

2. Terminology and assumptions. Two of the basic assumptions underlying bilevel programming are that full information is available to the players, and that cooperation is prohibited. This precludes the use of correlated strategies and side payments.

The following notation is used in the development.

Follower's Rational Reaction Set:

$$M(x) = \{y: y = \operatorname{argmax}\,[f(x, y): y \in Y, g(x, y) \geqq b]\}.$$

Inducible Region:

$$\mathbb{R} = \{(x, y): x \in X, y \in M(x)\}.$$

In order to ensure that (1a)-(1e) is well posed, we make the additional assumption that the feasible region (1b), (1d), and (1e) is nonempty and compact, and that for each decision taken by the leader, the follower has some room to respond. The rational reaction set $M(x)$ defines this response while the inducible region $\mathbb{R}$ represents the set over which the leader may optimize when given control of all the variables.

Even with the above assumptions, the BLPP may not have a solution. In particular, if $M(x)$ is not single-valued for all permissible x, the leader may not achieve his maximum payoff over $\mathbb{R}$. In order to avoid this situation, it will be assumed that $M(x)$ is a point-to-point map. Because a simple check is available to see if the solution to (1a)-(1e) is unique (see Bard and Falk 1982), we do not feel that this assumption is unduly restrictive. The problem actually solved by our algorithm is $\max\{F(x, y): (x, y) \in \mathbb{R}\}$ without the requirement that $M(x)$ be single-valued.

3. Methodology. Rather than working with (1a)-(1e) in its hierarchical form, we begin by converting it into a standard mathematical program. This is achieved by replacing the follower's problem (1c)-(1e) with his Kuhn-Tucker conditions, and giving control of all the variables to the leader (see Simaan and Cruz (1973)). For $X = \{x: x \geqq 0\}$, $Y = \{y: y \geqq 0\}$, and $m \equiv m_1$, we get

$$\text{(2a)} \qquad \max_x F(x, y) = c^1 x + c^2 y,$$

(2b) subject to $x^TQ_1+y^TQ_2+u^1B+u^2I=-c^3,$

(2c) $u^1(Ax+By-b)+u^2y=0,$

(2d) $Ax+By\geqq b,$

(2e) $x, y, u^1, u^2\geqq 0,$

where u^1 and u^2 are m- and n_2-dimensional Kuhn-Tucker multipliers, and I is an $n_2\times n_2$ identity matrix. Constraints (2b), (2c), and (2e) can be interpreted as an explicit representation of the inducible region. Thus, even for the linear/quadratic case, problem (1a)-(1e) is nonconvex and cannot necessarily be solved with a standard nonlinear programming code such as GRG2 (Lasdon et al. (1978)).

As suggested by Fortuny-Amat and McCarl, the basic idea of our algorithm is to suppress the complementarity term (2c) and solve the resulting linear program. At each iteration, a check is made to see if (2c) is satisfied. If so, the corresponding point is in the inducible region, and hence, is a potential solution to (1a)-(1e); if not, a branch and bound scheme is used to implicitly examine all combinations of complementary slackness. It should be mentioned that Fortuny-Amat and McCarl did not implement this scheme but took the more direct approach of replacing (2c) with the following set of inequalities: $u_i\leqq Mz_i$, $g_i\leqq M(1-z_i)$, where $z_i\in\{0,1\}$ for $i=1,\cdots,m+n_2$, and M is a sufficiently large constant. They then solved the resultant problem with a standard zero-one mixed integer code.

Before presenting the algorithm, we introduce some additional notation. Define $u\equiv(u^1,u^2)$, let $W=\{1,\cdots,m+n_2\}$ be the index set for the terms in (2c), and let $\underline{F}$ be the incumbent lower bound on the leader's objective function. At the kth level of the branch and bound tree we define a subset of indices $W_k\subseteq W$, and a path vector P_k (with $|W_k|$ nonzero components) corresponding to an assignment of either $u_i=0$ or $g_i=0$ for $i\in W_k$. Now let

$$S_k^+=\{i\colon i\in W_k \text{ and } u_i=0\},$$

$$S_k^-=\{i\colon i\in W_k \text{ and } g_i=0\},$$

$$S_k^0=\{i\colon i\notin W_k\}.$$

For $i\in S_k^0$, the variables u_i and g_i are free to assume any nonnegative values in the solution of (2a)-(2e) with (2c) omitted, so (2c) will not necessarily be satisfied.

Algorithm.

Step 0. (Initialization). Put $k=0$, $S_k^+=\varnothing$, $S_k^-=\varnothing$, $S_k^0=\{1,\cdots,m+n_2\}$, and $\underline{F}=-\infty$.

Step 1. (Iteration k). Set $u_i=0$ for $i\in S_k^+$ and $g_i=0$ for $i\in S_k^-$. Attempt to solve (2a)-(2e) without (2c). If the resultant subproblem is infeasible, go to Step 5; otherwise, put $k\leftarrow k+1$ and label the solution (x^k,y^k,u^k).

Step 2. (Fathoming). If $F(x^k,y^k)\leqq \underline{F}$, go to Step 5.

Step 3. (Branching). If $u_i^k\cdot g_i(x^k,y^k)=0$, $i=1,\cdots,m+n_2$, go to Step 4; otherwise, select i for which $u_i^k\cdot g_i(x^k,y^k)$ is largest and label it i_1. Put $S_k^+\leftarrow S_k^+\cup\{i_1\}$, $S_k^0\leftarrow S_k^0\backslash\{i_1\}$, $S_k^-\leftarrow S_k^-$, append i_1 to P_k, and go to Step 1.

Step 4. (Updating). $\underline{F}=F(x^k,y^k)$.

Step 5. (Backtracking). If no live node exists, go to Step 6. Otherwise branch to the newest live vertex and update S_k^+, S_k^-, S_k^0, and P_k as discussed below. Go to Step 1.

Step 6. (Termination). If $\underline{F} = -\infty$, there is no feasible solution to (1a)-(1e). Otherwise, declare the feasible point associated with $\underline{F}$ the optimal solution.

Step 1 is designed to find a new point that is potentially bilevel feasible. If no solution exists, or the solution does not offer an improvement over the incumbent (Step 2), the algorithm goes to Step 5 and backtracks. At Step 3, a check is made to determine if the complementary slackness conditions are satisfied. In practice, if $|u_i \cdot g_i| < 10^{-6}$, it is considered to be zero. Confirmation indicates that a feasible solution of the bilevel program has been found, and at Step 4, the lower bound on the leader's objective function is updated. Alternatively, if the complementary slackness conditions are not satisfied, the term with the largest product is used at Step 3 to provide the branching variable. Branching is always done on the Kuhn-Tucker multiplier.

At Step 5, the backtracking operation is performed. Note that a live node is one associated with a subproblem that has not yet been fathomed at either Step 1 due to infeasibility or at Step 2 due to bounding, and whose solution violates at least one complementary slackness condition. To facilitate bookkeeping, the path P_k in the branch and bound tree is represented by an l-dimensional vector, where l is the current depth of the tree. The order of the components of P_k is determined by their "level" in the tree. Indices only appear in P_k if they are in either S_k^+ or S_k^- with the entries underlined if they are in S_k^-. Because the algorithm always branches on a Kuhn-Tucker multiplier first, backtracking is accomplished by finding the rightmost nonunderlined component of P_k, underlining it, and erasing all entries to the right. The newly underlined entry is deleted from S_k^+ and added to S_k^-; the erased entries are deleted from S_k^- and added to S_k^0.

If we arrive at Step 6 and $\underline{F} = -\infty$, then we conclude that the original constraint region (1b), (1d), (1e) is empty. This will only be the case if the first subproblem at Step 1 is infeasible. Alternatively, the algorithm terminates with the incumbent whose optimality is now established.

PROPOSITION. *Under the uniqueness assumption associated with the rational reaction set $M(x)$, the algorithm terminates with the global optimum of the* BLPP (1a)-(1e).

Proof. The algorithm forces satisfaction of the complementary slackness conditions in problem (2a)-(2e), which is an equivalent representation of (1a)-(1e). By implicitly considering all combinations of $u \cdot g(x, y) = 0$ at Steps 3 and 5, the optimal solution cannot be overlooked. □

Example.

$$\max_x F(x, y) = 8x_1 + 4x_2 - 4y_1 + 40y_2 + 4y_3 \quad \text{where } y \text{ solves}$$

$$\max_y f(x, y) = -x_1 - 2x_2 - y_1 - y_2 - 2y_3,$$

$$\begin{array}{lrl} \text{subject to} & y_1 - y_2 - y_3 & \geqq -1, \\ & -2x_1 + y_1 - 2y_2 + 0.5y_3 & \geqq -1, \\ & -2x_2 - 2y_1 + y_2 + 0.5y_3 & \geqq -1, \\ & x \geqq 0, \quad y & \geqq 0. \end{array}$$

This example was taken from Candler and Townsley (1982). When it is rewritten in its equivalent form (2a)-(2e), six Kuhn-Tucker multipliers appear, implying that it may be necessary to solve as many as $2^7 - 1 = 127$ subproblems before terminating. In fact, the optimal solution was uncovered on the fourth iteration but was not confirmed until 10 subproblems were examined.

More specifically, after initializing the data, the algorithm finds a feasible solution to the Kuhn–Tucker representation with the complementary slackness conditions omitted, and proceeds to Step 3. The current point, $x^1=(0,0)$, $y^1=(1.5,\ 1.5,\ 1)$, $u^1=(0,0,0,1,1,2)$, with $F(x^1,y^1)=58$ does not satisfy complementarity so a branching variable is selected (u_6) and the index sets are updated, giving $S_1^+=\{6\}$, $S_1^-=\emptyset$, $S_1^0=\{1,2,3,4,5\}$, and $P_1=(6)$. In the next two iterations, the algorithm branches on u_5 and u_4, respectively. Now, three levels down in the tree, the current subproblem at Step 1 turns out to be infeasible, so the algorithm goes to Step 5 and backtracks. The index sets are $S_3^+=\{5,6\}$, $S_3^-=\{4\}$, and $S_3^0=\{1,2,3\}$, and $P_3=(6,5,\underline{4})$. Going to Step 1, a feasible solution is found that passes the test at Step 2 and satisfies the complementary slackness conditions at Step 3. Continuing at Step 4, $\underline{F}=29.2$. Backtracking at Step 5 yields $S_4^+=\{6\}$, $S_4^-=\{5\}$, and $S_4^0=\{1,2,3,4\}$, and $P_4=(6,\underline{5})$. Returning to Step 1, another feasible solution is found, but at Step 2, the value of the leader's objective function is less than the incumbent lower bound, so the algorithm goes to Step 5 and backtracks, giving $S_5^+=\emptyset$, $S_5^-=\{6\}$, $S_5^0=\{1,2,3,4,5\}$, and $P_5=(\underline{6})$. The calculations continue until no live vertices exist. The optimal solution is $x^*=(0,0.9)$, $y^*=(0,0.6,0.4)$, $u^*=(0,1,3,6,0,0)$ with $F^*=29.2$. The branch and bound tree for this example is shown in Fig. 1.

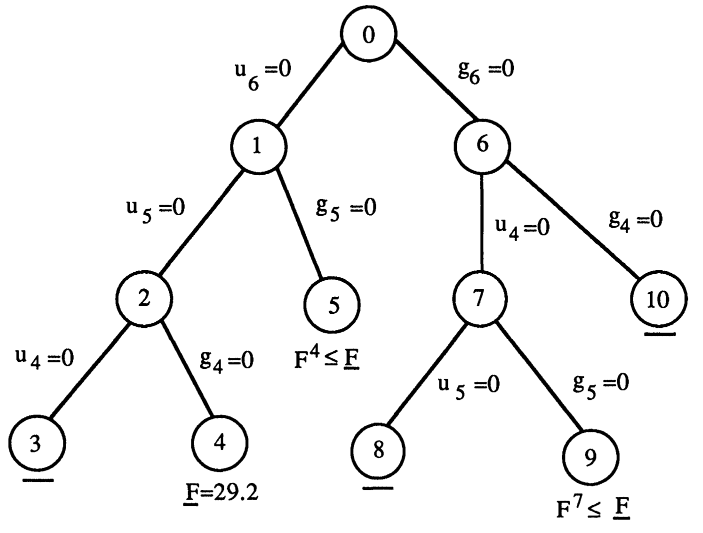


FIG. 1. *Branch and bound tree for example.*

By way of comparison, when this problem was solved with the separable programming approach of Bard and Falk (1982), the optimal solution was uncovered on the 51st iteration but not recognized until iteration 103. (Each iteration required the solution of a linear program in n_1+n_2+2m variables and $2(m+n_2)+1$ constraints.) This result typifies the relative performance of these two algorithms.

Finally, we note that the above procedure is considerably more general than that proposed by Bard (1988). Although both use branch and bound concepts, the latter takes an active set approach, adhering to the inducible region until a local optimum is found. This requires more bookkeeping, and will only succeed if the leader's objective function is strictly concave (for a maximization problem). Alternatively, our algorithm would easily solve this version of the BLPP if an appropriate nonlinear optimization code were used at Step 1 for the subproblems (see Edmunds (1988) for an implementation).

4. Computational experience. In order to test the efficiency of the algorithm, a series of problems was randomly generated and solved. For the primary cases reported, the coefficients of the A, B, Q_1, and Q_2 matrices ranged between -15 and 45 with approximately 25% of the entries being less than zero. Each matrix had a density of about 0.4. The coefficients of the two objective functions varied between -20 and 20 with approximately 50% being less than zero. The number of constraints in each problem was set at 0.4 times the total number of variables, and the right-hand side (RHS) values ranged between 0 and 50. The signs of the constraints had a 0.7 probability of being $\leqq$ and a 0.3 probability of being $\geqq$.

These settings are somewhat arbitrary, but were chosen to be consistent with previous work (e.g., see Bard (1983b), Bialas and Karwan (1984)). The advantage of having all coefficients of equal magnitude is that the resultant problems are almost always stable numerically. From a testing point of view, the matrix density factor plays an important role in generating random problems. Depending on the number of variables and constraints in the model, if this value is set too low a considerable amount of work may be required to assure that each realization is usable; i.e., unbiased and feasible with at least one nonzero element in each row and column. For our generator, a value of 0.4 was sufficient to guarantee usability in all but a few instances.

To complement the primary runs, additional testing was done on a subset of the original problems. In the first case, we investigated the relationship between algorithmic performance and the density of the A and B matrices. Here, the range of coefficient values remained the same. In the second case, an attempt was made to construct "ill-conditioned" problems by selectively generating coefficients on the order of 10^6.

All computations were performed on an IBM 3081-D using the VS Fortran compiler. As now coded, the subproblems encountered at Step 1 are solved with the linear programming (LP) subroutine library XMP (Marsten (1981)); however, any LP package including those based on interior point methods could be used. Multiplier values required to be zero on a given iteration are accommodated by fixing their upper and lower bounds at zero. Similarly, constraints required to be binding are satisfied by setting their slacks to zero. Both of these operations are easily handled in XMP by a subroutine call. All variable bounds are treated implicitly, so additional constraints are not needed in the formulation.

4.1. Results. Table 1 summarizes our computational experience with the algorithm for the all linear case; i.e., $f(x, y) = c^3 y$. Each entry represents the average of 10 randomly generated problems. In all, 110 problems were solved ranging in size from 40 to 100 variables, and 16 to 40 constraints. Performance measures include CPU time (seconds), the number of nodes in the branch and bound tree, and the node at which the optimal solution is found. Also, data for the largest and smallest search trees are given as a measure of variability.

As expected, the CPU time grows exponentially with the size of the problem, but more importantly, depends on the way the variables are partitioned between the players. As the number of variables controlled by the follower increases, the number of variables included in the branch and bound process increases along with the expected computational burden. Compare, for example, the two cases with 90 variables (and 36 constraints). On average, 81 additional CPU seconds are required for the case where $n_2 = 45$.

Also, as seen in Table 1, large differences in computational effort often result among problems of equivalent size. For the 100 variable case, 34 subproblems had to

TABLE 1
Computational results for the all linear case.

No. of variables (n_1+n_2)	Follower variables (n_2)	No. of consts. (m)	CPU time (sec)	Average no. of nodes	No. of nodes (range)	Optimal solution (node)
40	12	16	4.44	18	6-42	11
40	16	16	8.50	40	8-202	27
50	15	20	17.24	39	10-112	26
50	20	20	16.46	35	18-124	23
50	25	20	32.39	73	16-218	46
50	30	20	179.01	447	30-1250	391
70	28	28	106.99	96	32-270	67
70	35	28	122.26	106	26-246	84
90	36	36	352.37	138	30-384	81
90	45	36	433.14	185	48-374	122
100	40	40	294.22	159	34-476	120

be solved at one extreme and 476 at the other. The corresponding CPU times were 85 seconds and 804 seconds, respectively. In general, the optimum is not uncovered until 60 to 70% of the realized tree is examined. This implies that if the algorithm is stopped prematurely the best solution might be missed. A final point to be made about the empirical results is that about 45% of the nodes in the search tree are fathomed due to infeasibility, and rarely (only 5% of the time) due to a solution being in the inducible region. As discussed in § 4.2, this is due in large part to the branching rule employed at Step 3. The remaining 50% are fathomed at Step 2 when the relaxed solution is less than or equal to the incumbent.

In order to see if the algorithm performed any differently when the follower's objective function contained quadratic terms, the problem sets were rerun for the case where $f(x, y) = c^3y + x^TQ_1y + \frac{1}{2}y^TQ_2y$. In the actual implementation, Q_2 is coded as a lower triangular matrix to facilitate data input. This format eliminates duplicate entries.

The second set of results is presented in Table 2 where little if any significant difference can be seen when compared to the results in Table 1. Nevertheless, the algorithm does seem to take a bit longer to converge when the quadratic terms are

TABLE 2
Computational results for the linear/quadratic case.

No. of variables (n_1+n_2)	Follower variables (n_2)	No. of consts. (m)	CPU time (sec)	Average no. of nodes	No. of nodes (range)	Optimal solution (node)
40	12	16	4.83	20	6-55	13
40	16	16	9.01	45	7-238	30
50	15	20	16.79	37	9-114	25
50	20	20	18.71	41	22-139	31
50	25	20	38.28	82	19-205	48
50	30	20	192.37	482	30-1163	372
70	28	28	116.99	104	38-299	75
70	35	28	112.65	104	32-281	71
90	36	36	393.21	143	28-407	89
90	45	36	451.78	202	53-392	130
100	40	40	363.93	178	38-511	132

added. Only problem sets 3 and 8 show an improvement. In general, this slight degradation in performance was traced to the fact that (2b) was more easily satisfied when terms in x and y were present. Compared to the all linear case, fathoming at Step 1 due to infeasibility was not as likely to occur at the early iterations. This produced slightly larger trees.

The next set of runs was designed to see how algorithmic performance varied with the density of the A and B matrices. In order to limit the computational effort of this phase of the analysis, two representative problems sets were singled out for investigation, and only the linear case was considered. The first problem set is characterized by parameter values $(n, m, n_1, n_2) = (50, 20, 25, 25)$, and the second set by $(70, 28, 35, 35)$. The results for density factors of 0.2, 0.3, and 0.4 are reported in Table 3. All entries represent an average of 10 cases. As the density is reduced from 0.4 to 0.3, the average CPU time stays about the same while the number of nodes in the search tree increases modestly. For a density factor of 0.2, however, a significant drop in CPU time is observed. In the first case, this is accompanied by an incresse in the average size of the tree from 73 nodes to 100 nodes, and in the second case by a decrease from 106 nodes to 82 nodes.

TABLE 3
Results for different matrix densities for all linear case.

No. of variables (n_1+n_2)	Follower variables (n_2)	No. of consts. (m)	CPU time (sec)	Average no. of nodes	No. of nodes (range)	Optimal solution (node)
Matrix density = 0.4						
50	25	20	32.39	73	16-218	46
70	35	28	122.26	106	26-246	84
Matrix density = 0.3						
50	25	20	32.11	86	16-208	68
70	35	28	135.52	119	26-332	95
Matrix density = 0.2						
50	25	20	22.29	100	22-382	59
70	35	28	67.84	82	16-236	55

Although no definitive conclusions should be drawn from these findings, it would be fair to say that a positive correlation exists between the overall density of the problem and the average time spent on each subproblem. This might be accounted for by a combination of factors. First, as the density decreases, the average time XMP takes to solve each subproblem decreases as well. Second, lower densities increase the likelihood that subproblems at a given level in the tree will be infeasibility. This implies that fewer LPs will have to be solved.

Finally, it should be mentioned that when the density factor was set to 0.1, our random problem generator failed to produce any problems in the 50 variable case that did not have either a null row or null column (most of these problems were feasible, though). In the 70 variable case, about 1 in 20 randomly generated problems were usable.

The last set of runs was aimed at determining how well the algorithm performs on problems with widely varying coefficients. In all, eight different scenarios were examined for the 50 variable case. The density factor was held constant at 0.4 and no quadratic terms were included in the follower's objective function $f(x, y)$. Each scenario was characterized by coefficient values randomly selected from one of two ranges. For the cost coefficients, the ranges were $[-20, 20]$ and $[-10^6, 10^6]$; for the A and B matrices, the ranges were $[-15, 45]$ and $[-2.5\times10^5, 7.5\times10^5]$; and for the RHSs, the ranges were $[0, 50]$ and $[0, 10^6]$.

The results are displayed in Table 4. Each row represents an average of 10 cases, with the data in the first row taken from Table 1. Average CPU times range from 23.4 seconds to 78.5 seconds, but no clear patterns emerge. However, the algorithm does seem to be slightly less efficient when the problems are ill-conditioned. The only real point worth making is that XMP had no trouble solving any of the subproblems.

TABLE 4
Results for ill-conditioned problems.
($n = 50$, $m = 20$, $n_1 = n_2 = 25$, density $= 0.4$).

Range of coefficients†			CPU	Average	No. of	Optimal
Matrices	Cost	RHS	time	no. of nodes	nodes (range)	solution (node)
[−15, 45]	[−20, 20]	[0, 50]	32.4	73	16-218	46
[−15, 45]	[−20, 20]	[0, e6]	43.6	119	14-284	109
[−15, 45]	[−e6, e6]	[0, 50]	78.5	207	60-420	97
[−15, 45]	[−e6, e6]	[0, e6]	34.6	100	30-216	73
[−0.25e6, 0.75e6]	[−20, 20]	[0, 50]	45.9	102	26-242	83
[−0.25e6, 0.75e6]	[−20, 20]	[0, e6]	51.8	141	18-264	110
[−0.25e6, 0.75e6]	[−e6, e6]	[0, 50]	23.4	56	24-82	41
[−0.25e6, 0.75e6]	[−e6, e6]	[0, e6]	47.6	132	52-194	102

† The notation e6 denotes 10^6.

It is interesting to compare the above findings with those reported by others. Although Fortuny-Amat and McCarl (1981) did not seriously test their scheme, it is an easy matter to obtain an assessment. After investigating a few 20 variable problems with ZOOM (a zero-one, mixed integer version of XMP), we found that the accompanying run times and search trees were 10 to 100 times larger than ours. The reasons for this were twofold. First, their approach leads to problems with an additional $2(m + n_2)$ constraints and $m + n_2$ variables; second, ZOOM has its own built in branching rules that are not necessarily "optimal" for BLPPs (the same could be said for any commercial mixed integer code).

Bialas and Karwan (1984) report results for both their "Kth-best" algorithm and their Parametric Complementary Pivot (PCP) approach. (While the former did not fare too well and will not be discussed, the latter should really be considered a heuristic because convergence is not guaranteed. In addition, it is limited by the requirement that the leader's objective function coefficients associated with the follower's variables be nonnegative; i.e., $c^2 \geqq 0$.) The largest problems they solved contained 50 variables, with 20 being controlled by the follower. The number of constraints was fixed at 0.4 times the number of variables. Each data set contained five problems, and all computations were done on a CDC Cyber 174 using a Fortran IV code.

Table 5 presents a comparison of the PCP algorithm, our branch and bound scheme, and the separable approach of Bard and Falk. The latter is roughly equivalent to the zero-one formulation of Fortuny-Amat and McCarl in that both approaches lead to problems of nearly identical size and structure. Note that Bard's (1983b) grid search algorithm is not discussed because it only works for BLPPs whose solutions are known to be Pareto-optimal. As can be seen, the first two algorithms are on equal footing with respect to CPU time, and outdistance the third by more than an order of

TABLE 5
Comparison with other algorithms.

No. of variables (n_1+n_2)	Follower variables (n_2)	PCP CPU time (sec)†	B&B CPU time (sec)‡	Separable CPU time (sec)‡
40	12	4.46	4.44	76.25
40	16	11.23	8.48	145.93
50	15	16.48	17.24	298.31
50	20	16.52	16.46	344.77

† Average of 5 problems; CDC Cyber 174.
‡ Average of 10 problems; IBM 3081-D.

magnitude. Note that the Cyber 174 and the IBM 3081-D each perform about 1.7 million floating point operations per second when solving dense systems of linear equations using the LINPACK software (Dongarra (1986)). Nevertheless, the experimental nature of the codes, coupled with the fact that different test problems and different machines were used in the analyses, strongly argue against drawing all but the most tentative conclusions from the data in Table 5. To credibly determine the relative performance of each algorithm, a much more deliberate experimental design would have to be established. At a minimum, it would be necessary to examine a wide variety of problems under identical conditions using the same LP package at each stage in the computations.

4.2. Alternatives explored during testing. In the actual implementation, whenever a feasible solution is found at Step 1, the accompanying basis is stored in XMP format and used as the starting point for the next subproblem. Because this subproblem only differs from its parent by a single constraint (at the next level down just one additional u_i or g_i is set to zero), only a few pivots are normally required to regain feasibility. This was seen to provide a relative advantage when compared to two other procedures tested for maintaining a starting basis. For the first alternative, the previous basis whether feasible or not was used to initiate the next subproblem. For small formulations the results suggested no significant difference in CPU time; but for problems with 50 or more variables, a slight degradation in performance was observed. The second alternative used the basis associated with the last point found in the inducible region. If none existed, the last basis was used. This procedure yielded CPU times two to three times larger than the others and hence is not recommended. In general, the ability to quickly find a feasible solution is the key. On average, the latter procedure spent more than half its time in phase I of the simplex algorithm.

An important determinant of computational efficiency is the rule for selecting the branching variable at Step 3. The rule chosen branches on the Kuhn-Tucker (KT) multiplier associated with the largest complementarity term $u_i \cdot g_i$. Another rule that was examined, primarily because it worked well when nonlinear BLPPs were solved with an active set strategy (see Bard (1988)), proved to be noncompetitive. In this case, the selection is made by finding the surface on which F increases most rapidly; i.e., by solving

$$(3)\qquad \max_i \left[\nabla F(x^k, y^k) \cdot \nabla g_i(x^k, y^k) / \|\nabla F(x^k, y^k)\| \, \|\nabla g_i(x^k, y^k)\|\right]$$

where ∇ is the gradient operator and $\|\cdot\|$ is the Euclidean norm. Using this rule and branching on g produced large increases in CPU time; branching on u led to some relative improvement but anywhere from 50 to 100 additional subproblems had to be solved. These results were observed for moderate sized problems where $n_1 = 15$, $n_2 = 10$, and $m = 10$. Note that for linear/quadratic BLPPs, (3) automatically establishes the branching order. This is probably the reason why it did not work well.

After testing several other rules, it became evident that it is never advantageous to branch on g first. In general, when this strategy is used there is little early fathoming due to infeasibility so the left-hand side of the tree (denoted by $g_i = 0$) grows rapidly. Alternatively, by forcing the KT multipliers to zero, one of two situations quickly arises: either the gradient equations (2b) become infeasible or a point in the inducible region emerges. For this reason, we limited our testing to the following branching rules:

(1) KT multiplier associated with largest $u \cdot g$ product,
(2) KT multiplier with largest value,
(3) KT multiplier with smallest value,
(4) KT multiplier associated with largest g,
(5) KT multiplier associated with smallest g,
(6) KT multiplier associated with smallest $u \cdot g$ product,

where the first is the procedure implemented. Table 6 displays our findings when each of these rules is applied to the original 50 variable problem set. The data represent an average of 10 cases for the all linear version of problem (1a)-(1e).

TABLE 6
Comparison of results for different branching rules.
($n = 50$, $m = 10$, $n_1 = 25$, $n_2 = 25$, density $= 0.4$)

Rule no.	CPU time	Average no. of nodes	No. of nodes (range)	Optimal solution (node)
1	32.39	73	16-218	46
2	38.91	78	15-225	49
3	98.44	127	20-404	83
4	72.12	106	18-374	71
5	47.66	82	16-271	57
6	68.83	92	18-326	64

As shown in Table 6, none of the variants performed as well as the first rule. Increases in average CPU time of roughly 5 to 66 seconds or 12 to 230% can be observed. In general, those rules that give priority to large values of u seem to do better; the first two rules exhibit almost identical performance.

One possible explanation for these results centers on (2b). Here, the second set of KT multipliers u^2, corresponding to the nonnegativity constraints on the y variables, can be viewed as slacks. At the early stages of the computations these variables tend to take on large values and are hence selected for branching. When this happens, (2b) is usually more difficult to satisfy so fathoming due to infeasibility is more prevalent.

By implication, a rule that gives priority to the u^2 variables might also be a good choice, but this remains to be tested. Finally, note that the results accompanying this phase of the analysis might very well have been different had quadratic terms been included in follower's objective function.

5. Summary and conclusions. Experience has shown that even for the simplest of formulations, the bilevel programming problem is inherently difficult to solve. The branch and bound algorithm developed in this paper attempts to exploit some of the special structure in the problem, and in so doing, is able to achieve rapid convergence. Linear problems with up to 100 variables and 40 constraints can be solved in less than 300 seconds on average. After thorough testing, the algorithm's performance and robustness are seen to compare favorably with virtually all contenders. A scarcity of data on other algorithms, though, limits the strength of this assertion. More and better controlled experiments are needed before any final conclusions can be drawn.

Nevertheless, one of the main advantages of the branch and bound approach is that it is quite general. Although our analysis centered on the linear/quadratic formulation, solutions to the nonlinear problem can be readily obtained by substituting a more general code such as GRG2 for the XMP library. The algorithm is valid for all functional forms, as well as the case where more than one follower is present. Convergence to the global optimum, though, can only be guaranteed when certain convexity and separable properties hold.

REFERENCES

E. AIYOSHI AND K. SHIMIZU (1981), *Hierarchical decentralized systems and its new solution by a barrier method*, IEEE Trans. Systems, Man, Cybernetics, 11, pp. 444-449.

J. F. BARD (1988), *Convex two-level optimization*, Math. Programming, 40, pp. 15-27.

——— (1983a), *Coordination of a multidivisional firm through two levels of management*, Omega, 11, pp. 457-468.

——— (1983b), *An efficient point algorithm for a linear two-stage optimization problem*, Oper. Res., 31, pp. 670-684.

J. F. BARD AND J. E. FALK (1982), *An explicit solution to the multi-level programming problem*, Comput. Oper. Res., 9, pp. 77-100.

T. BASAR AND H. SELBUZ (1979), *Closed loop Stackelberg strategies with applications in optimal control of multilevel systems*, IEEE Trans. Automat. Control, 24, pp. 166-178.

W. F. BIALAS AND M. H. KARWAN (1984), *Two-level linear programming*, Management Sci., 30, pp. 1004-1020.

W. CANDLER AND R. TOWNSLEY (1982), *A linear two-level programming problem*, Comput. Oper. Res., 9, pp. 59-76.

J. J. DONGARRA (1986), *Performance of various computers using standard linear equations software in a Fortran environment*, Technical Memorandum No. 23, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL.

T. A. EDMUNDS (1988), *Algorithms for nonlinear bilevel mathematical programs*, Ph.D thesis, Department of Mechanical Engineering, University of Texas, Austin, TX.

J. FORTUNY-AMAT AND B. MCCARL (1981), *A representation and economic interpretation of a two-level programming problem*, J. Oper. Res. Soc., 32, pp. 783-792.

B. F. GARDNER, JR. AND J. B. CRUZ, JR. (1978), *Feedback Stackelberg strategy for M-level hierarchical games*, IEEE Trans. Automat. Control, 23, pp. 489-491.

R. G. JEROSLOW (1985), *The polynomial hierarchy and a simple model for competitive analysis*, Math. Programming, 32, pp. 146-164.

L. S. LASDON, A. D. WARREN, A. JAIN, AND M. RATNER (1978), *Design and testing of a generalized reduced gradient code for nonlinear programming*, ACM Trans. Math. Software, 4, pp. 34-50.

R. E. MARSTEN (1981), *The design of the* XMP *linear programming library*, ACM Trans. Math. Software, 7, pp. 481-497.

M. SIMAAN AND J. B. CRUZ, JR. (1973), *On the Stackelberg strategy in nonzero-sum games*, J. Optim. Theory Appl., 11, pp. 533-555.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 2, pp. 293-310, March 1990

A DETERMINISTIC APPROXIMATION OF DIFFUSION EQUATIONS USING PARTICLES*

PIERRE DEGOND† AND FRANCISCO-JOSÉ MUSTIELES‡

Abstract. A new particle method adapted to the simulation of convection-diffusion problems is presented. The method relies on the definition of a convective field associated with the heat operator, which allows the convection of the particles in a deterministic way.

No rigorous error analysis is given but, instead, a detailed numerical study of the influence of the parameters is presented. This study is performed on the heat equation, then applied to a Fokker-Planck model arising in the kinetic theory of plasma physics. Finally, an extension of the method to other problems is given.

Key words. diffusion equations, particle method, Fokker-Planck equation

AMS(MOS) subject classifications. 35K05, 65C20, 65M25, 82A45

1. Introduction. Particle methods have been used for a long time to give a numerical solution of purely convective problems, such as the incompressible Euler equation in fluid mechanics, or the Vlasov equation in plasma physics (cf. the reviews of Harlow [9], Leonard [12], [13], and the books of Hockney and Eastwood [11] and Birdsall and Langdon [1]). Indeed, they give rise to accurate solutions with very few numerical diffusion, and they are extremely simple to interpret in a physical sense.

It thus became interesting to wonder if they could be adapted to slightly diffusive problems (e.g., the Navier-Stokes equation in fluid mechanics, or the equations of the kinetic theory in presence of scattering processes, such as the neutron transport equation, the Boltzmann equation of gas dynamics or of semiconductors, and the Fokker-Planck equation of plasma physics).

The most famous particle method used to solve these problems is the Monte Carlo method: the diffusive term is modeled by random motions of the particles according to a suitable probability law (cf. Chorin [3] for the Navier-Stokes equation and [5], [11] for the equations of the kinetic theory). The success of this method is mainly due to the fact that the dynamics of the numerical particles is a reproduction of the dynamics of the physical ones. However, from a numerical viewpoint, the random choices introduce a very large amount of noise leading somehow to inaccurate computations.

Recently there has appeared a new particle method that relies on a deterministic treatment of the diffusive term and that could be expected to give more accurate results. This method has been proposed by Cottet, Mas-Gallic [2], and Mas-Gallic and Raviart [15] for the Navier-Stokes equations, and adapted by Mas-Gallic to the equations of the kinetic theory [14]. It makes use of a supplementary degree of freedom associated with each particle: its "weight." The convective (Euler or Vlasov) part of the equation is modeled by the convection of the particles, while the diffusive part (heat or Boltzmann operator) is taken into account by the variation of the weights, through a finite-difference-like approximation of the diffusion operator, in which the particles act as meshpoints. Recent tests (cf. [16], [4]) have shown that this "weighted particle method" could provide quite accurate results.

* Received by the editors November 18, 1987; accepted for publication (in revised form) November 21, 1988.

† Centre de Mathématiques Appliquées, Ecole Polytechnique, 91128 Palaiseau, France.
‡ Ecole des Mines de Madrid, Rio Rosas 21, 28003 Madrid, Spain.

However, one of the main drawbacks of this method is that the positions of the particles do not have any physical meaning. Indeed, the diffusion processes do not modify them but rather the weights, and thus, the information on the solution cannot be recovered from the positions of the particles alone; the complete set of positions and weights has to be considered.

Therefore, it seemed interesting to derive an intermediate method between the Monte Carlo and the weighted particle method: a method that would rely on a deterministic treatment of the diffusion operator (to provide an accurate approximation) but that would model the diffusion process by the motion of the particles in some physically relevant way. The purpose of this paper is to present such a method for the case of second-order diffusion operators. The case of an integral operator (Boltzmann or transport) is much more difficult, and has not been solved yet.

For the heat operator, this method relies on the interpretation of Fick's law in a deterministic way. Let $u(x, t)$ be a solution of the heat equation. Then Fick's law states that the flux of u is proportional to $-\nabla u$. A particle approximation of the heat equation thus will be achieved, if we can move the particles in the direction of $-\nabla u$ (the exact magnitude will be given later). Since ∇u itself depends on the particle distribution, a smoothing is necessary to recover a smooth flux ∇u. As announced, this method is deterministic (no random choices are involved), and leads to an actual motion of the particles.

The idea of using Fick's law to derive deterministic Lagrangian schemes for the heat equation has previously been used by Fronteau and Combis [6] and Grmela, Fronteau, and Tellez-Arenas [8]. However, their method differs in the computation of the flux ∇u. For this purpose, they use finite-difference methods on the moving grid. In this respect, their method is a moving grid method rather than a particle method. The use of a smoothing procedure to compute ∇u allows us to get rid of the grid, and gives more flexibility. Recently, using our method, Hermeline [10] has obtained interesting results on an alpha particle transport problem, showing that the present method can be useful for real physical problems.

No error analysis is yet available for this method. But the purpose of this paper is to present numerical results that give evidence that this method converges to the solution of the heat equation. Practical considerations on the reliability of this method for plasma physical simulations are also detailed.

This paper is organized as follows. In § 2 we give a detailed presentation of the method, followed by some heuristic considerations concerning the accuracy of the scheme in § 3. In § 4 we display numerical tests performed on the case of the one-dimensional heat equation. In § 5 two-dimensional results on a Fokker–Planck model are presented. A conclusion is drawn in § 6.

2. Introducing the method. We consider a solution $u(x, t)$, $x \in \mathbb{R}^n$, $t > 0$ of the following heat equation on $\mathbb{R}^n$:

$$\begin{aligned} &\frac{\partial u}{\partial t} - \nabla \cdot (S(x, t) \cdot \nabla u) = 0, \\ &u(x, 0) = u_0(x), \end{aligned} \tag{2.1}$$

where $S(x, t)$ is a $n \times n$ positive definite matrix. Following a classical idea in the physics of diffusion processes, we can rewrite (2.1) into a couple of equations:

$$\frac{\partial u}{\partial t} + \nabla \cdot j = 0 \quad \text{(conservation equation)}, \tag{2.2}$$

$$j(x,t)=-S(x,t)\cdot\nabla u(x,t)\quad\text{(Fick's law)}.\tag{2.3}$$

Such a formulation is also the basis of the method of Fronteau and Combis (see [6], [8]), and more classically, of the mixed finite-element method.

Then we rewrite the conservation equation (2.2) as a convection equation by

$$\frac{\partial u}{\partial t}+\nabla\cdot(A(x,t)u)=0,\tag{2.4}$$

where, from (2.3), the vector field $A(x, t)$ is given by

$$A(x,t)u(x,t)=j(x,t)=-S(x,t)\cdot\nabla u(x,t);\tag{2.5}$$

thus

$$A(x,t)=-S(x,t)\cdot\nabla u(x,t)/u(x,t).\tag{2.6}$$

Now, if we forget (2.6) temporarily, and if we assume that $A(x, t)$ is given and known, then (2.4) is simply a convection equation, for which the particle approximation is classical (cf. Raviart [17]). Let $u_h^0(x)$ be the particle approximation of the initial data $u_0(x)$:

$$u_h^0(x)=\sum_j \alpha_j\delta(x-x_j^0)\simeq u_0(x).\tag{2.7}$$

Then an approximate solution $u_h(x, t)$ of (2.4) is found by letting

$$u_h(x,t)=\sum_j \alpha_j\delta(x-X_j(t)),\tag{2.8}$$

where $X_j(t)$ is the characteristic curve associated with the vector field $A(x, t)$, issued from the point x_j^0:

$$\frac{dX_j}{dt}=A(X_j(t),t)\qquad X_j(0)=x_j^0.\tag{2.9}$$

Then, going back to equation (2.6), we see that $A(x, t)$ is not known, but depends on the solution u itself. Furthermore, replacing u in (2.6) by its approximation u_h given by (2.8) is meaningless, because the ratio of linear combinations of Dirac masses cannot be defined in a proper way. The idea is thus, to introduce a smoothed approximation $u_h^\varepsilon(x, t)$ by

$$u_h^\varepsilon(x,t)=(u_h*\zeta_\varepsilon)(x,t)=\sum_j \alpha_j\zeta_\varepsilon(x-X_j(t)),\tag{2.10}$$

where the cutoff function $\zeta_\varepsilon(x)$ is such that

$$\zeta_\varepsilon(x)=\zeta(x/\varepsilon)/\varepsilon^n,\qquad \int\zeta(x)\,dx=1.\tag{2.11}$$

Then, gradients and ratios involving u_h^ε can be computed in a proper way and an approximation of A can be given as

$$A_h^\varepsilon(x,t)=-S(x,t)\cdot\nabla u_h^\varepsilon(x,t)/u_h^\varepsilon(x,t).\tag{2.12}$$

Now the particle approximation of (2.1) can be given completely:

$$u(x,t)\simeq u_h(x,t)=\sum_i \alpha_i\delta(x-X_i(t)),$$

where $(X_i(t))$ is the solution of the following system of differential equations:

$$\frac{dX_i}{dt}=-\frac{S(X_i(t),t)\cdot\sum_j \alpha_j\nabla\zeta_\varepsilon(X_i(t)-X_j(t))}{\sum_j \alpha_j\zeta_\varepsilon(X_i(t)-X_j(t))},\tag{2.13}$$
$$X_i(0)=x_i^0,$$

and the initial positions x_i^0 and the weights α_j are chosen to give an approximation of the initial data according to (2.7). As we have mentioned previously, the weights α_j do not vary in time. If they are initialized equally they will remain equal, and all the information will be carried by the positions of the particles.

3. Various comments.

3.1. Initialization. One frequently used initialization algorithm is to choose an equal volume initialization (cf. Raviart [17]). For that purpose, the computational domain is divided into cells B_i of equal volume ω, centered at x_i and we let

$$u_h^0(x) = \sum_{j=1}^{N} \omega u^0(x_j)\delta(x-x_j). \tag{3.1}$$

This leads to a nonconstant weight $\alpha_j = \omega u^0(x_j)$ and is therefore more suitable to weighted particle methods, in which weights are already subject to time variations.

Similarly, if source terms or zeroth-order terms were present in the equation, the weights would no longer be constant in time, and the initialization (3.1) would be perfectly suitable.

Therefore, for the above-presented method, an equiweighted initialization is preferable. One way to achieve this in one dimension for a positive initial data $u_0(x)$ is to use the change of variable:

$$y(x) = \int_{-\infty}^{x} u_0(s)\, ds \cdot \Big/ \int_{-\infty}^{\infty} u_0(s)\, ds, \tag{3.2}$$

which is a one-to-one, onto, smooth mapping from $\mathbb{R}$ to $[0, 1]$, of inverse $x(y)$. Thus, for any compactly supported smooth function φ on $\mathbb{R}$, we obtain

$$\int_{-\infty}^{\infty} u_0(x)\varphi(x)\, dx = \int_0^1 \varphi(x(y))\, dy \cdot \int_{-\infty}^{\infty} u_0(s)\, ds. \tag{3.3}$$

The equiweighted quadrature of (3.3) leads to

$$\int_{-\infty}^{\infty} u_0(x)\varphi(x)\, dx \simeq \sum_{j=1}^{N} \frac{1}{N}\varphi\left(x\left(\left(j-\frac{1}{2}\right)N^{-1}\right)\right) \cdot \int_{-\infty}^{\infty} u_0(s)\, ds, \tag{3.4}$$

which is equivalent to the following particle approximation:

$$u_0(x) \simeq u_h^0(x) = \left(N^{-1}\int_{-\infty}^{\infty} u_0(s)\, ds\right) \cdot \sum_{j=1}^{N} \delta(x-x_j), \tag{3.5}$$

where x_j is given by

$$x_j = x(N^{-1}(j-\tfrac{1}{2})), \tag{3.6}$$

or equivalently,

$$\int_{-\infty}^{x_j} u_0(s)\, ds = N^{-1}\int_{-\infty}^{\infty} u_0(x)\, dx \cdot \left(j-\frac{1}{2}\right). \tag{3.7}$$

Extensions of this algorithm to $\mathbb{R}^d$ can be given easily (cf., e.g., [11]).

3.2. Alternate schemes. We can imagine other ways to recover a smoothed convection field $A_h^\varepsilon(x, t)$ from the particle distribution $u_h(x, t)$. For instance, we can choose different cutoff functions for the approximations of ∇u and of u. Indeed, let $\mu_\varepsilon(x)$ be an approximation of the gradient of the Dirac mass:

$$\begin{gathered}\mu_\varepsilon(x)=\frac{1}{\varepsilon^{n+1}}\mu\left(\frac{x}{\varepsilon}\right), \qquad \mu(x)=(\mu_1(x),\cdots,\mu_n(x)),\\ \int \mu_i(x)\,dx=0, \qquad \int x_j\mu_i(x)\,dx=-\delta_{ij}.\end{gathered} \tag{3.8}$$

Then, we can take

$$A_h^\varepsilon(x,t)=-\frac{S(x,t)\cdot\sum_j \alpha_j\mu_\varepsilon(x-X_j(t))}{\sum_j \alpha_j\zeta_\varepsilon(x-X_j(t))}. \tag{3.9}$$

Another method could be based on the cloud in cell (C.I.C.) methodology (cf. [9], [11]). If u was known on a fixed regular grid, then (2.6) would be easy to compute by finite differencing. The C.I.C. methodology consists in using assignment and interpolation procedures to connect the grid quantities with the particle quantities.

3.3. Accuracy of the method. The error analysis for this method has not yet been performed. However, we think that it is possible to prove that classical error estimates for particle methods apply to this new method, at least in any compact region where u does not vanish. Indeed the error can be written:

$$\begin{aligned}e(x,t)&=u(x,t)-\sum_j \alpha_j\zeta_\varepsilon(x-X_j^h(t))\\ &=u(x,t)-\sum_j \alpha_j\zeta_\varepsilon(x-X_j(t))+\sum_j \alpha_j[\zeta_\varepsilon(x-X_j(t))-\zeta_\varepsilon(x-X_j^h(t))],\end{aligned} \tag{3.10}$$

where $X_j(t)$ is the solution of equation (2.9) and $X_j^h(t)$ is the solution of system (2.13). The first term in formula (3.10) is classically estimated in L^∞ norm by $C(\varepsilon^k+(h/\varepsilon)^m)$ where h is the mean interparticle distance, m is related to the smoothness of the cutoff ζ, and k is related to the higher-order vanishing moment of ζ (cf. Raviart [17]). The second term is estimated as soon as the difference $X_j(t)-X_j^h(t)$ is estimated. Again, in the region where u does not vanish (say $u\geqq\alpha>0$), this estimation reduces to a "convolution and quadrature" estimate for the flux $A(x,t)$, in the same spirit as Raviart [17]. Of course, the argument has to be bootstrapped, and the rigorous proof will be given in a forthcoming paper.

Therefore, we expect the error to be of the form

$$\varepsilon^k+\left(\frac{h}{\varepsilon}\right)^m. \tag{3.11}$$

From a practical viewpoint, this has several consequences. Indeed, for a given interparticle distance h (fixed by the number of particles N), there is an optimal value of ε for which the error is minimal. Besides, for a given ε, the error deteriorates as h grows. However, the motion of particles under diffusion is generally an expansion and thus, h grows with time, leading to unacceptably large errors after a sufficiently long simulation time. A remedy against this would be to allow ε to vary in order to achieve the minimum value of the error (3.11) at any time. In § 4, we will present a method to perform this, even locally, by considering a "local" interparticle distance h.

However, this method displays special behaviors that other particle methods do not present. The most obvious one is that the method is nonlinear. Indeed, for two different initial data $u^1(x)$ and $u^2(x)$, the positions of the particles $X_j^1(t)$ and $X_j^2(t)$

will be different, and different from the positions $X_j^{1+2}(t)$ of the particles associated with the initial data $u^1(x)+u^2(x)$. Thus we will never get

$$u_h^{1+2}(x,t)=u_h^1(x,t)+u_h^2(x,t),$$

as should be expected from the linearity of the equation.

Another special behavior is displayed whenever the exact solution u vanishes, or at least becomes very small. In this case, the convection field $A(x,t)$ becomes very high or even infinite. Of course, such a behavior cannot be followed by the numerical solution, for which the velocity, for stability reasons, must remain finite, and bounded by a maximum velocity A^*. Thus, instead of a diffusion at an infinite velocity as should be expected from a linear diffusion equation, the method generates diffusion at a finite velocity A^*, with creation of a diffusion front (as in some nonlinear diffusion phenomena). This front creation may be the source of instabilities.

3.4. Time discretization and stability. Of course, the time differential equation (2.13) must be solved numerically, and throughout the following tests, an explicit Euler scheme has been used. Therefore, the question of stability arises. We have numerically verified a stability constraint of the type

$$\Delta t < c\varepsilon^2, \tag{3.12}$$

with c of the order of unity, but such a relation still has to be proved.

4. One-dimensional tests on the heat equation.

4.1. Comparisons of the exact and approximate solution. In this section, we present numerical tests of the scheme (2.13) performed on the one-dimensional heat equation:

$$\frac{\partial u}{\partial t}-\frac{\partial^2 u}{\partial x^2}=0, \qquad x\in\mathbb{R}, \quad t\geqq 0. \tag{4.1}$$

A first series of tests have been performed for a positive solution $u(x,t)$. The initial data has been chosen to be the characteristic function of the $[-\frac{1}{2},\frac{1}{2}]$ interval. In Figs. 1(a) and 1(b), snapshots of the exact and approximate solutions are displayed for different times. For these numerical experiments, two different types of cutoff functions $\zeta(x)$ have been used. In Fig. 1(a), an order 2B-spline has been employed:

$$\zeta_1(x)=\begin{cases}\frac{3}{4}-x^2, & 0\leqq|x|\leqq\frac{1}{2},\\ \frac{1}{2}(\frac{3}{2}-|x|)^2, & \frac{1}{2}\leqq|x|\leqq\frac{3}{2},\end{cases} \tag{4.2}$$

whereas in Fig. 1(b), a super-Gaussian cutoff has been considered:

$$\zeta_2(x)=\Pi^{-1/2}(\tfrac{3}{2}-x^2)\exp(-x^2). \tag{4.3}$$

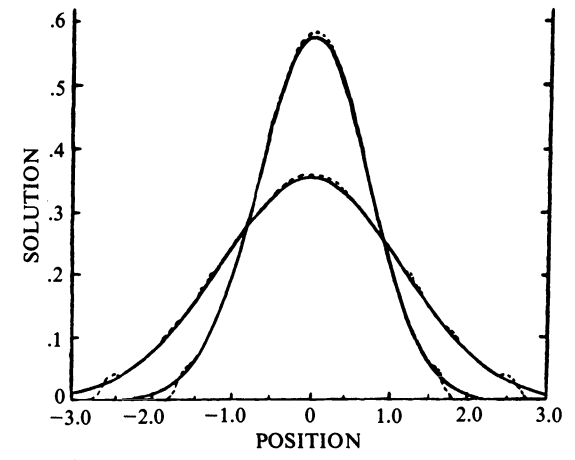
The order 2 B-spline is only once continuously differentiable and satisfies

$$\int x^2\zeta_1(x)\,dx\neq 0, \tag{4.4}$$

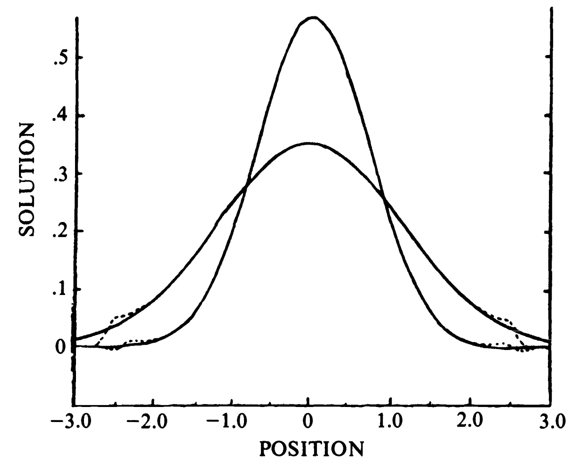
whereas the super-Gaussian cutoff is infinitely differentiable and has a first nonvanishing moment of higher order:

$$\int x^2\zeta_2(x)\,dx=0, \qquad \int x^4\zeta_2(x)\,dx\neq 0. \tag{4.5}$$

Thus both m and k in the error formula (3.11) are higher for ζ_2, leading to a more precise smoothing procedure.



(a) Order 2 B-spline cutoff. Timestep $DT = 0.01$.



(b) Super-Gaussian cutoff. Timestep $DT = 0.02$.

FIG. 1. *Heat equation solutions at times $T = 0.2$ and $T = 0.6$. Regularization parameter: $\varepsilon = 0.2$. Number of particles: $N = 100$. Solid line: exact solutions. Dashed line: calculated solutions.*

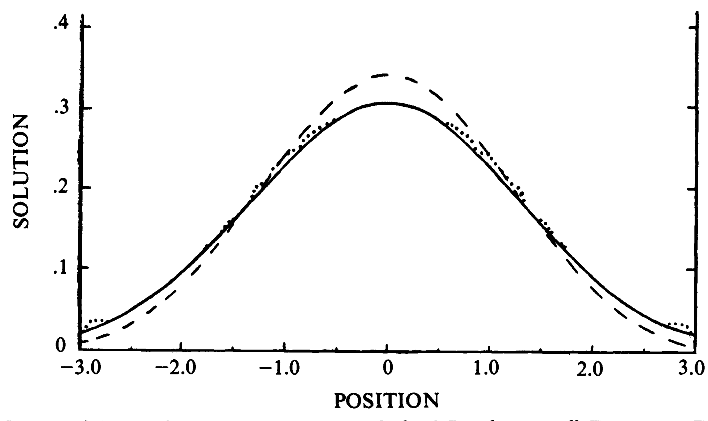


FIG. 2. *Influence of the regularization parameter ε. Order 2 B-spline cutoff. Timestep $DT = 0.02$. Number of particles: $N = 100$. Time: $T = 0.8$. Exact solution: solid line. Calculated solution ($\varepsilon = 2$): dashed line. Calculated solution ($\varepsilon = 0.2$): dotted line.*

Indeed, in Figs. 1(a) and 1(b) we see that the use of the super-Gaussian cutoff leads to a better approximation; however, it seems to be more sensitive to the formation of the diffusion front in the external areas, where the solution is small. Furthermore, the B-spline is compactly supported; this fact can be exploited to reduce the computational cost, which is not possible with the super-Gaussian.

Figure 2 shows that, as expected, too small values of ε lead to parasitic oscillations while too large values of ε generate unacceptable smoothing errors. An average number of particles per cutoff of 10–15 seems to lead to the most accurate results.

Finally, Fig. 3 shows that the nonobservation of a stability requirement such as (3.12) results in nonphysical oscillations.

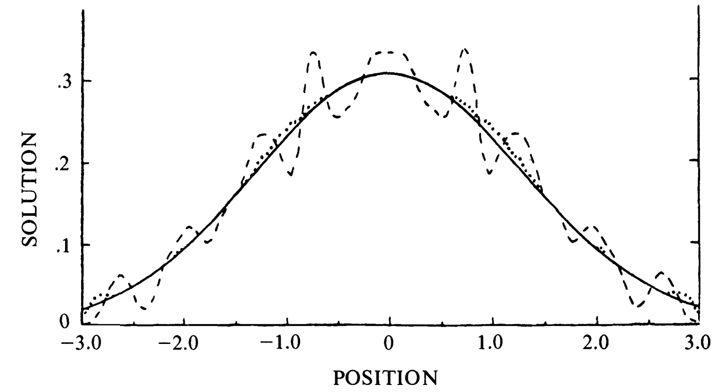


FIG. 3. *Influence of the timestep DT. Order* 2 B-*spline cutoff. Regularization parameter*: $\varepsilon = 0.2$. *Number of particles*: $N = 100$. *Time*: $T = 0.8$. *Exact solution*: *solid line. Calculated solution* ($DT = 0.04$): *dashed line. Calculated solution* ($DT = 0.02$): *dotted line.*

A second test has been made with a solution of nonconstant sign. This is a more severe test, since the approximation is expected to be bad in the area where the solution vanishes (cf. § 3.3). The initial data and the exact and approximate solutions are given on Fig. 4 after 80 iterations. Though the approximation is not as good as in the case of a positive solution, it remains stable, and leads to a yet qualitatively reliable solution. For all these tests, an equiweighted initialization was used.

4.2. Analysis of the convection field. The whole method relies on the approximation of the convection field (2.6) by the formula (2.12), and thus, on the quality of the smoothing (or interpolation) procedures, allowing the recovery of point values from the particle distribution (2.8). Thus, a detailed study of this approximation has been made, in the case of a typical Gaussian profile:

$$u(x) = \Pi^{-1/2} \exp(-x^2). \tag{4.6}$$

Function (4.6) has first been approximated by the particle distribution arising from an equiweighted initialization procedure (3.5):

$$u_h(x) = N^{-1} \sum_{j=1}^{N} \delta(x - x_j), \qquad \int_{-\infty}^{x_j} \exp(-s^2)\, ds = \Pi^{1/2} N^{-1}\left(j - \frac{1}{2}\right). \tag{4.7}$$

Then, a smoothing procedure has been applied to (4.7) to compute

$$u_h^\varepsilon(x) = N^{-1} \sum_{j=1}^{N} \zeta_\varepsilon(x - x_j), \tag{4.8}$$

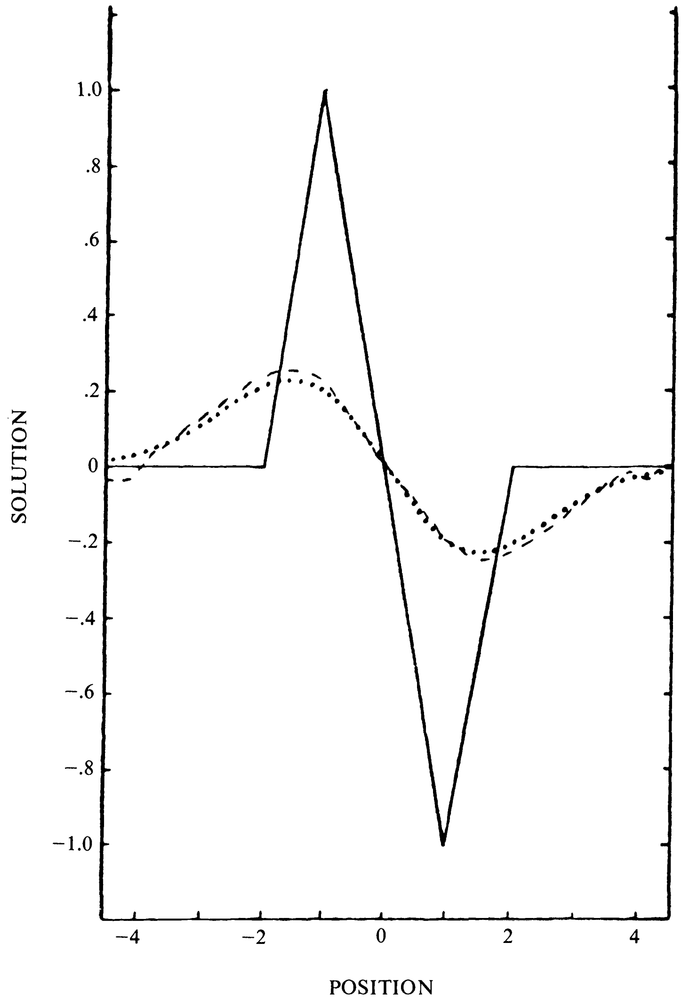


FIG. 4. *Heat equation solutions with nonconstant sign. Number of particles*: $N = 100$. *Regularization parameter*: $\varepsilon = 0.3$. *Timestep*: $DT = 0.01$. *Initial data*: *solid line. Exact solution at time* $T = 0.8$: *dotted line. Calculated solution at time* $T = 0.8$: *dashed line.*

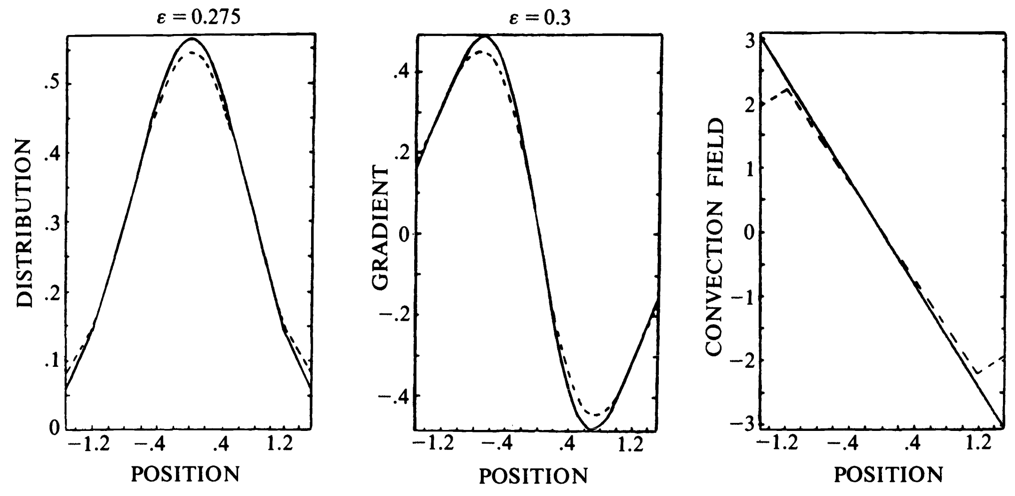
$$\frac{du_h^\varepsilon}{dx} = N^{-1} \sum_{j=1}^{N} \zeta_\varepsilon'(x - x_j), \tag{4.9}$$

$$A_h^\varepsilon(x) = \frac{du_h^\varepsilon}{dx}(x)/u_h^\varepsilon(x). \tag{4.10}$$

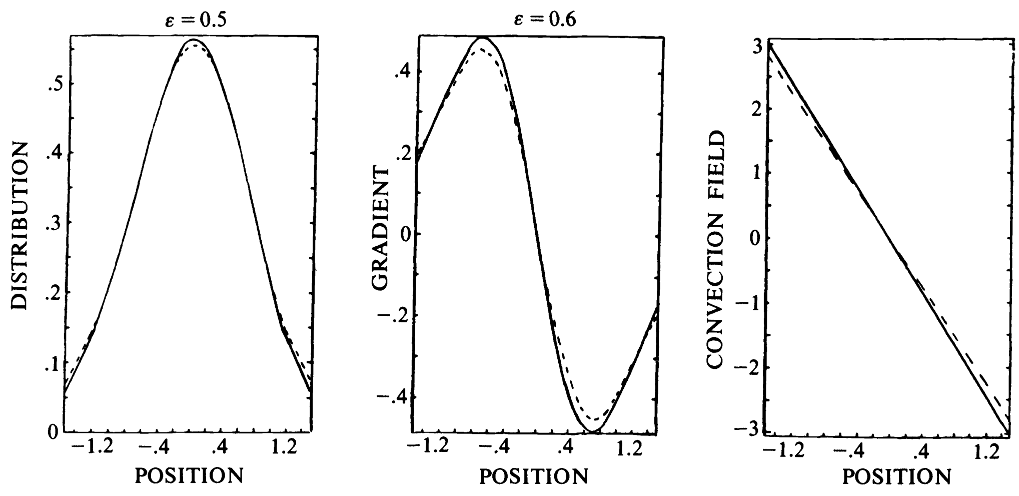
These values have been compared with the exact ones, given from (4.6):

$$u'(x) = -2\Pi^{-1/2} x \exp(-x^2), \qquad A(x) = -2x.$$

In Figs. 5(a) and 5(b) these comparisons are plotted for an order 2 B-spline cutoff and a super-Gaussian cutoff. For each $u_h^\varepsilon(x)$, $u_h'^\varepsilon(x)$, the "best" value of ε has been chosen (on empirical grounds). We see that it is not the same for u and u', and for each choice of the cutoff. A larger ε is needed for u' than for u, and this is understandable,



(a) Order 2 B-spline cutoff.



(b) Super-Gaussian cutoff.

FIG. 5. *Convection field approximation. Number of particles*: $N = 32$. *Left figures*: *approximation of* $u(x) = e^{-x^2}/\sqrt{\pi}$. *Central figures*: *approximation of* $u'(x)$. *Right figures*: *approximation of* $u'(x)/u(x)$. *Exact functions*: *solid line. Approximate functions*: *dashed line.*

since the differentiation of ζ_ε lowers the accuracy of the quadrature error, which must be balanced by choosing a larger ε.

These figures essentially show that an accurate approximation of the convection field is difficult, especially where u is small. Thus, the convection of the outermost particles is very poorly accurate, and this creates the diffusion front that was apparent in Figs. 1(a) and 1(b). Of course, the error on the outermost particles propagates from particle to particle inside the domain, as time proceeds, and this is one of the major drawbacks of this method. However, for the super-Gaussian cutoff, this error is small, which explains the better results obtained in this case.

In order to get a better improvement of the method, a smoothing procedure with variable ε has been tried. Indeed, the smoothing is worse in regions where $u(x)$ is smaller, that is, where the (local) interparticle distance h is larger. If we assume, based on [17], that the local error is of the form

$$\varepsilon^k + \left(\frac{h}{\varepsilon}\right)^m, \tag{4.11}$$

we see that in these regions the quadrature error $(h/\varepsilon)^m$ may be larger by several orders of magnitude than the convolution error ε^k. Thus, if we could increase ε in these regions, this would lower the local error. This suggests binding ε and h by a relation that, based on (4.11), must be algebraic:

$$\varepsilon = Ch^\alpha, \qquad 0 < \alpha < 1. \tag{4.12}$$

Then, for each particle j, an estimate of h_j is given by assuming that the weight $(1/N)$ must be an approximation of $h_j \cdot u(x_j)$. Indeed, we let

$$h_j = (Nu(x_j))^{-1}, \qquad \varepsilon_j = Ch_j^\alpha, \tag{4.13}$$

and the smoothing u_h^ε is now written

$$u_h^\varepsilon(x) = \sum_j \frac{1}{N} \zeta_{\varepsilon_j}(x - x_j). \tag{4.14}$$

The biggest problem to solve is how to find suitable values of C and α. In the absence of a mathematical theory for (4.12), only a numerical study is possible, which makes the method very empirical and problem dependent.

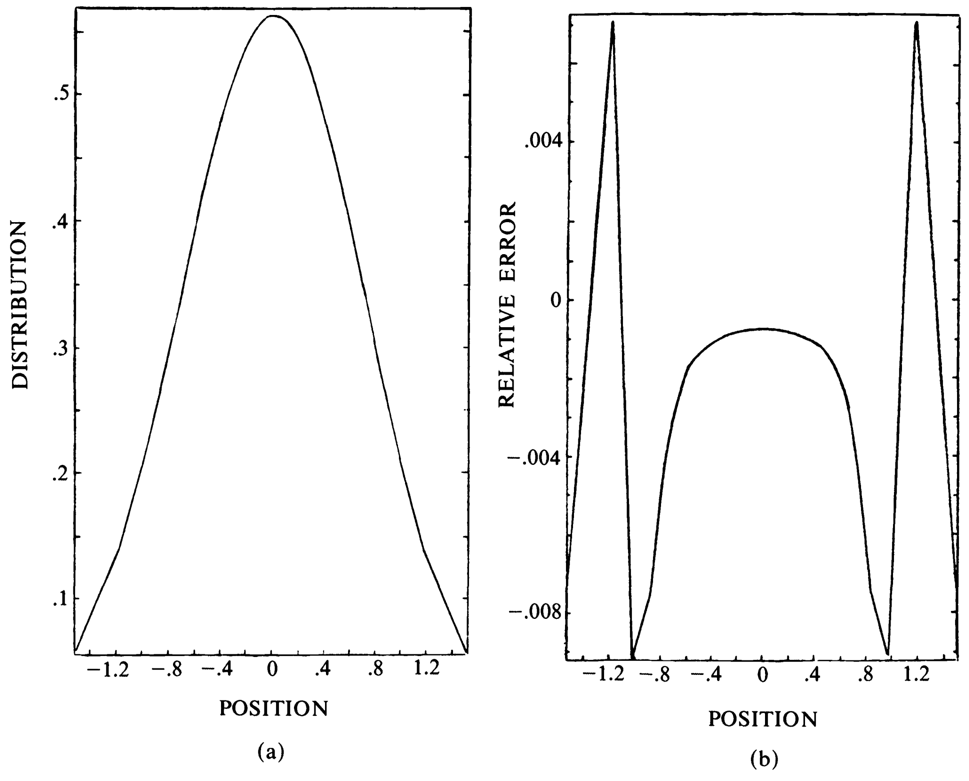


FIG. 6.1. *Approximation of the distribution function* (a) *and relative error* (b) *with variable* ε. *Gaussian cutoff. Number of particles*: $N = 32$. $C = 0.49$. $\alpha = 0.6$.

We have tested in a rather systematical way all the reasonable values of C and α, in order to give the best approximation of (4.6) and its derivative, with formula (4.14). Of course, different values of C and α must be chosen to minimize the error on u and u'.

However, values of C and α can be designed so as to lead to an error on u less than 1 percent all over the domain, with a Gaussian cutoff and only 32 particles (cf. Fig. 6.1). For the derivative $u'(x)$ things are worse: we must face the alternative that either we get a very good accuracy inside the domain (again less than 1 percent), but with a 25 percent error on the outermost particles, or we get a more homogeneous accuracy of about 5 percent all over the domain (cf. Fig. 6.2). Of course, for the outermost particles, the interparticle distance is not a well-defined concept (logically it should be equal to infinity). Thus, the improvement is not as large as expected for

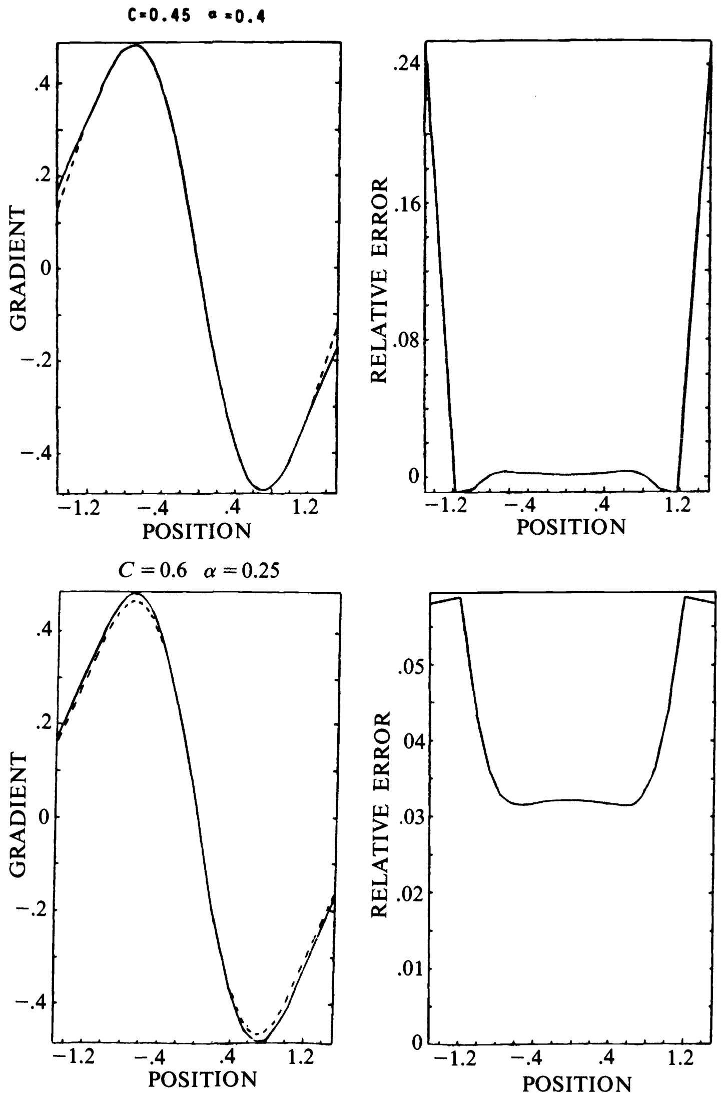


FIG. 6.2. *Approximation of the gradient and relative error with variable ε. Gaussian cutoff. Number of particles: $N = 32$. Exact functions: solid line. Approximate functions: dashed line.*

the region near the outermost particles. Then, again, the error is likely to propagate from particle to particle inside the domain, as time proceeds. For this reason, and due to its complexity, this method with a variable ε has not been retained for further computations.

5. Two-dimensional tests on a Fokker–Planck model.

5.1. Introduction. We intend to model a two-dimensional infinite and homogeneous sample of plasma or semiconductor, submitted to an external constant electric field E. The kinetic theory then provides the distribution function of electrons that, in this case, is a function of the velocity $v \in \mathbb{R}^2$ and the time t alone, $f(v, t)$. In our model, the equation for $f(v, t)$ is written:

$$(5.1) \qquad \frac{\partial f}{\partial t} + \sum_{i=1}^{2} E_i \frac{\partial f}{\partial v_i} = \frac{1}{T} \sum_{i=1}^{2} \frac{\partial}{\partial v_i}(v_i f) + \sum_{i=1}^{2} \frac{\partial^2 f}{\partial v_i^2}.$$

The left-hand side represents the acceleration of the particles under the electric field, while the right-hand side is a Fokker–Planck model used to describe the interactions of the particles with a medium at a temperature T. Of course, this model is very crude, but it retains many features of the kinetic models, and will provide a more realistic test problem for our method.

We will consider a two-dimensional velocity space $v = (v_1, v_2)$, and initialize with a Maxwellian distribution at temperature T_0:

$$(5.2) \qquad f_0(v) = M_{T_0}(v) = (2\Pi T_0)^{-1} \exp(-|v|^2/2T_0), \qquad |v|^2 = v_1^2 + v_2^2.$$

The solution of (5.1) can then be computed analytically, and in particular, the stationary solution is given by a displaced Maxwellian at temperature T:

$$(5.3) \qquad f_\infty(v) = (2\Pi T)^{-1} \exp(-|v - ET|^2/2T).$$

If $E = 0$, the equation models the relaxation of the initial distribution function M_{T_0} toward the stationary solution of the Fokker–Planck operator M_T.

5.2. The numerical scheme. Equation (5.1) is a convection-diffusion problem. Using the method of § 2 it can be written in the form of a convection equation:

$$(5.4) \qquad \frac{\partial f}{\partial t} + \nabla_v . (A(v, t) f) = 0$$

with

$$(5.5) \qquad A(v, t) = E - \frac{v}{T} - \frac{\nabla_v f}{f},$$

where $\nabla_v .$ and ∇_v denote, respectively, the divergence and the gradient operators with respect to the variable v.

Then, let f_h be a particle approximation of f:

$$(5.6) \qquad f_h(v, t) = N^{-1} \sum_{i=1}^{N} \delta(v - V_i(t)).$$

Following the ideas of § 2, we get a particle discretization of (5.1) by letting

$$(5.7) \qquad \frac{dV_i}{dt} = E - \frac{1}{T} V_i(t) - \frac{\sum_{j=1}^{N} \nabla \zeta_\varepsilon(V_i(t) - V_j(t))}{\sum_{j=1}^{N} \zeta_\varepsilon(V_i(t) - V_j(t))}.$$

An alternate scheme can be derived by noting that

$$-\frac{v}{T}=\nabla_v M_T(v)/M_T(v), \tag{5.8}$$

where $M_T(v)$ is given by (5.2). Then let a particle approximation of M_T be defined by

$$M_T(v)\simeq N^{-1}\sum_{j=1}^{N}\delta(v-W_j), \tag{5.9}$$

where W_j are fixed "Maxwellian" particles. Then we get an approximation of (5.8) by letting

$$-\frac{v}{T}\simeq\frac{\sum_{j=1}^{N}\nabla\zeta_\varepsilon(v-W_j)}{\sum_{j=1}^{N}\zeta_\varepsilon(c-W_j)}.$$

This leads to the following scheme:

$$\frac{dV_i}{dt}=E+\frac{\sum_j\nabla\zeta_\varepsilon(V_i(t)-W_j)}{\sum_j\zeta_\varepsilon(V_i(t)-W_j)}-\frac{\sum_j\nabla\zeta_\varepsilon(V_i(t)-V_j(t))}{\sum_j\zeta_\varepsilon(V_i(t)-V_j(t))}. \tag{5.10}$$

From a physical viewpoint, (5.10) can be interpreted as a dynamics of particles subject to a mutually repelling force, and to an attracting force, from the fixed particles W_i. In some sense, this gives a pictorial idea of the relaxation mechanism: when $E=0$, the particles will have a tendency to forget their initial distribution (due to the repelling force) and to "fall" into the attractor holes W_i, which will make them represent the equilibrium distribution function. However, as we will see, the numerics do not follow this nice physical picture, and the scheme (5.7) is better.

5.3. Numerical experiments. As a first test, we have investigated the relaxation process. In this case, a zero electric field has been considered and the relaxation of the initial distribution M_{T_0} toward the equilibrium Maxwellian M_T has been investigated. The contour lines of the calculated solution have been plotted, after a sufficiently long time, so that a steady state has been reached (Fig. 7). Two different cutoffs have been used, and for each of them, the two schemes (5.7) and (5.10) have been tested.

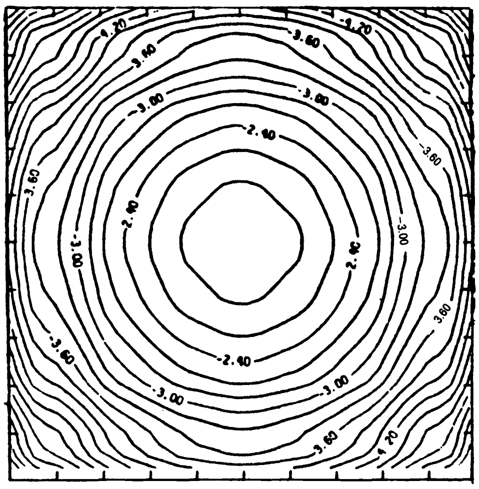
From Fig. 7 it appears that there is no real advantage to using (5.10) rather than (5.7), despite its nice physical interpretation. Since (5.10) is computationally more costly, it does not seem to be worthwhile using it.

Besides, the use of the super-Gaussian cutoff leads to a considerable improvement of the method. As we have seen in § 4.2, this is mainly due to the better accuracy of the smoothing procedure.

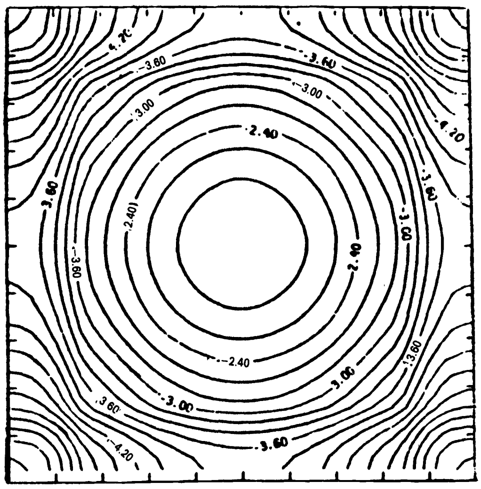
In a second numerical experiment, an electric field has been applied, and again, contour lines of the stationary computed solution have been drawn (cf. Fig. 8). It appears more clearly that the scheme (5.7) gives better results than (5.10).

As a general conclusion of these experiments, we can say that this method is very sensitive to the smoothing procedure used to recover a smooth convection field from the distribution of particles. However, the use of smooth high-order cutoffs such as the super-Gaussian leads to reliable results in the model cases that we have investigated. Whether it would give as good results in real physical situations is an open question. We believe that more accurate smoothing procedures have to be designed and tested for that method to be usable in physical codes.

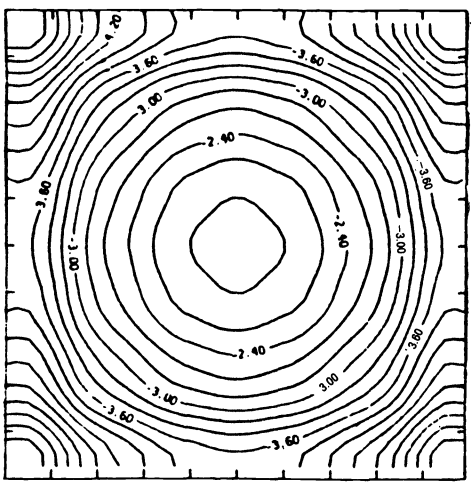
6. Conclusion. In this conclusion, we would like to emphasize that this method can be used in any evolution problem that can be written in a natural way as a



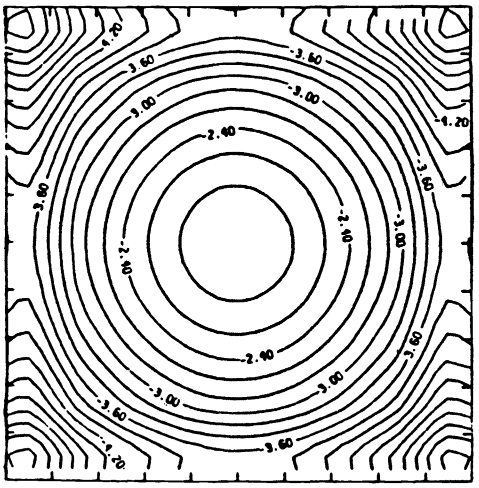
Order 2 B-spline cutoff Scheme (5.7)



Super-Gaussian cutoff Scheme (5.7)

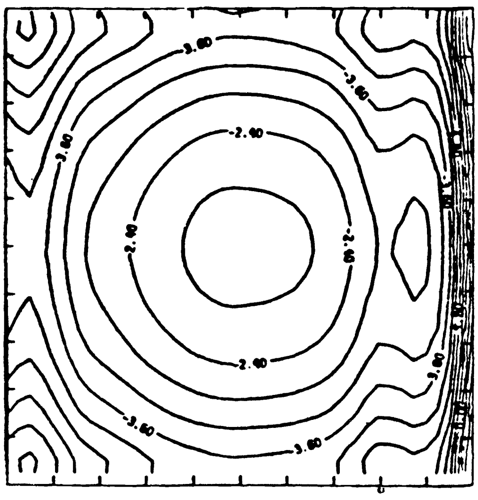


Order 2 B-spline cutoff Scheme (5.10)

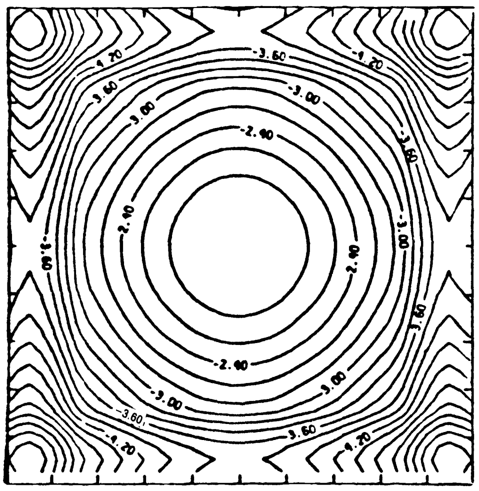


Super-Gaussian cutoff Scheme (5.10)

FIG. 7. *Stationary solution of the Fokker–Planck equation. Electric field*: $E=0$. *Initial temperature*: $T_0=1.1$. *Equilibrium temperature*: $T=1$. *Number of particles*: $N=1024$.



Scheme (5.10)



Scheme (5.7)

FIG. 8. *Stationary solution of the Fokker-Planck equation. Electric field:* $E = 0.5$. *Super-Gaussian cutoff. Same values of* T, T_0, *and* N *as in Fig.* 7.

convection equation. As an illustration, and though we have not performed any numerical test yet, we show that first-order systems can be simulated by such a method.

Let us recall that weighted particle methods have already been applied to first-order systems (cf. Mas-Gallic and Raviart [15]) and to the system of gas dynamics (cf. the smoothed particle hydrodynamics of Gingold and Monaghan [7]). These methods consist in picking a particular velocity among all the set of characteristic velocities available (it is usually the velocity of the fluid); the convection of the other characteristic fields is performed by a suitable variation of the weights.

The ideas exposed in this paper apply to first-order systems and allow all of the characteristic fields of the system to be modeled by a convection of particles, which is more natural. Indeed, let $u(x, t) = (u_1, \cdots, u_m)(x, t)$ be a solution of the following first-order system:

$$\frac{\partial u_\alpha}{\partial t} + \sum_{\beta,k} \frac{\partial}{\partial x_k}(a^k_{\alpha\beta}(x, t)u_\beta) = 0, \qquad \alpha = 1, \cdots, m, \tag{6.1}$$

where $x \in \mathbb{R}^n$, and $a^k_{\alpha,\beta}(x, t)$ are real functions (possibly depending on u). System (4.1) can be rewritten as a system of convection equations:

$$\frac{\partial u_\alpha}{\partial t} + \sum_k \frac{\partial}{\partial x_k}(A^k_\alpha(x, t)u_\alpha(x, t)) = 0, \qquad \alpha = 1, \cdots, m, \tag{6.2}$$

where

$$A^k_\alpha(x, t) = \left(\sum_\beta a^k_{\alpha\beta}(x, t)u_\beta(x, t)\right) \Big/ u_\alpha(x, t). \tag{6.3}$$

Now we allocate a different set of particles $\{X^\alpha_{i_\alpha}(t), i_\alpha = 1, \cdots, N_\alpha\}$ to each component u_α of the vector field u. And we define a particle approximation of u_α as

$$u_\alpha(x, t) \simeq \sum_{i_\alpha=1}^{N_\alpha} \omega^\alpha_{i_\alpha} \delta(x - X^\alpha_{i_\alpha}(t)). \tag{6.4}$$

Then, each set of particles $\{X^\alpha_{i_\alpha}\}$ is moved according to its associated convection field (4.3). Now since the convection field itself depends on the solution u, an interpolation procedure is again necessary to recover a smoothed approximation of the convection field. Thus the scheme can be written

$$\frac{dX^\alpha_{i_\alpha}}{dt} = A_{\varepsilon,\alpha}(X^\alpha_{i_\alpha}(t), t) = {}^t(A^1_{\varepsilon,\alpha}, \cdots, A^n_{\varepsilon,\alpha}), \tag{6.5}$$

where

$$A^k_{\varepsilon,\alpha}(x, t) = \frac{\sum_\beta a^k_{\alpha\beta}(x, t) \sum_{j_\beta=1}^{N_\beta} \omega^\beta_{j_\beta} \zeta_\varepsilon(x - X^\beta_{j_\beta}(t))}{\sum_{j_\alpha=1}^{N_\alpha} \omega^\alpha_{j_\alpha} \zeta_\varepsilon(x - X^\alpha_{j_\alpha}(t))}, \tag{6.6}$$

where ζ is an appropriate cutoff function. Using these ideas, it seems possible to approximate first-order systems by purely deterministic convections of particles.

Now, we summarize the conclusions of the numerical study presented in this paper.

First, the numerical results seem to evidence that this method is consistent and convergent, and this is an encouragement to try to obtain a convergence proof and error estimates for this method.

Second, to the question "is it worthwhile using this method," we can give a twofold answer. In the standard cases, such as the Navier-Stokes equation or the heat equation, it seems that this method may be less accurate and more expensive than the standard finite-difference or finite-element methods, or (at high Reynolds number) than the weighted particle method. Now, for problems arising in kinetic theory, where the high dimensionality of the problem does not allow the use of standard methods, such a method can be useful, at least to give a qualitative behavior of the solution.

In any case, a sharper study of the interpolation procedure used to recover a smooth convection field from the particle distribution would certainly lead to a considerable improvement of the accuracy of the method.

Furthermore, to be able to cover most cases in kinetic theory, the method must be extended to Boltzmann integral operators. Such an extension presents major difficulties that we have not yet been able to overcome.

REFERENCES

[1] C. K. Birdsall and A. B. Langdon, *Plasma Physics via Computer Simulations*, McGraw-Hill, New York, 1985.

[2] G. H. Cottet and S. Mas-Gallic, *A particle method to solve transport-diffusion equations, Part* 1: *The linear case*, Report No. 115, Ecole Polytechnique, Palaiseau, France, 1983.

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

[4] B. Niclot, P. Degond, and F. Poupaud, *Deterministic particle simulations of the Boltzmann equation of semiconductors*, J. Comput. Phys., 78 (1988), pp. 313-349.

[5] J. J. Duderstadt and M. R. Martin, *Transport Theory*, John Wiley, New York, 1979.

[6] J. Fronteau and P. Combis, *A Lie admissible method of integration of Fokker-Planck equations with non-linear coefficients (exact and numerical solutions)*, Hadronic J., 7 (1984), pp. 911-930.

[7] J. J. Monaghan and R. A. Gingold, *Shock simulation by the particle method* SPH, J. Comput. Phys., 52 (1983), pp. 374-389.

[8] M. Grmela, J. Fronteau, and A. Tellez-Arenas, *Inverse Liouville problem*, Hadronic J., 3 (1980), pp. 1209-1241.

[9] F. H. Harlow, *The particle-in-cell method for fluid dynamics*, in Methods in Computational Physics, B. Alder, S. Fernbach, and M. Rotenberg, eds., Academic Press, New York, 1964, Vol. 3.

[10] F. Hermeline, *A deterministic particle method for transport diffusion equations: Application to the Fokker-Planck equation*, J. Comput. Phys., 82 (1989), pp. 122-146.

[11] R. W. Hockney and J. W. Eastwood, *Computer Simulations Using Particles*, McGraw-Hill, New York, 1981.

[12] A. Leonard, *Vortex methods for flow simulations*, J. Comput. Phys., 37 (1980), pp. 289-335.

[13] ———, *Computing three-dimensional incompressible flows with vortex elements*, Ann. Rev. Fluid Mech., 17 (1985), pp. 523-559.

[14] S. Mas-Gallic, *A deterministic particle method for the linearized Boltzmann equation*, Transport Theory Statist. Phys., 16 (1987), pp. 855-887.

[15] S. Mas-Gallic and P. A. Raviart, *Particle approximation of convection-diffusion problems*, Report, University of Paris 6, Paris, France, 1985.

[16] S. Mas-Gallic and F. Poupaud, *Approximation of the transport equation by a weighted particle method*, Transport Theory Statist. Phys., to appear.

[17] P. A. Raviart, *An analysis of particle methods*, in Numerical Methods in Fluid Dynamics, F. Brezzi, ed., Lecture Notes in Mathematics 1127, Springer-Verlag, Berlin, New York, 1985.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 2, pp. 311-342, March 1990

FINITE-ELEMENT PRECONDITIONING FOR PSEUDOSPECTRAL SOLUTIONS OF ELLIPTIC PROBLEMS*

M. O. DEVILLE† AND E. H. MUND‡

Abstract. A preconditioning technique for pseudospectral solutions of elliptic problems based on quadrangular finite-element algorithms is analyzed, which exhibits excellent convergence properties. The pseudospectral technique is implemented through a collocation grid based on Gauss–Lobatto quadrature nodes associated to the Jacobi orthogonal polynomials. Various types of basis functions are used in the finite-element preconditioner (i.e., low-order Lagrange or cubic Hermite elements). Dirichlet and Neumann problems are investigated in one- and two-space dimensions. Numerical results show that the eigenvalue spectrum of the iteration matrix is inside the unit circle and even, close to zero for a wide range of operators. This property ensures convergence until roundoff error level in a few iterations. The differences between finite-element and finite-difference preconditioning are analyzed. Finally, the application of the algorithm to a problem exhibiting geometric induced singularities is discussed.

Key words. finite element, pseudospectral method, eigenvalue analysis

AMS(MOS) subject classifications. 65N30, 65N35, 65B05

1. Introduction.

Some fields in physics are particularly demanding in terms of numerical simulations of engineering problems or natural phenomena. Among them, numerical fluid mechanics has emerged as a major discipline concerned with applications in aerodynamics, thermal convection, direct simulation of turbulence, meteorology, etc.

When looking back over the last decades, we are striken by the importance of low-order classical methods such as finite differences (FD) or finite elements (FE) that have pervaded almost all applications of computational fluid mechanics. Other strategies, however, consist in using higher-order schemes that have accomplished breakthroughs in recent years. In that respect spectral techniques offer attractive properties such as exponential rate of convergence, reduced numerical dispersion and ability of treating high Reynolds number flows. Spectral schemes are based on several projection methods: Tau, Galerkin, and collocation [2]. In recent years, much experience relied on the Tau approach for simple geometries. However, the need of simulation tools able to treat general geometries leads to the choice of the pseudospectral approach that is carried out completely in the physical domain. The pseudospectral (or orthogonal collocation) technique does not suffer from the restrictions imposed by the Tau method. We may note that within the pseudospectral context, nonconstant coefficients in the partial differential equations are easily handled. Another comment comes from the fact that the numerical integration of the Navier–Stokes equations calls for efficient solvers of simpler problems such as Helmholtz or Poisson equations. This is the reason we investigate general second-order elliptic problems with nonconstant coefficients.

The pseudospectral approach imposes the mathematical problem to be solved exactly on a set of discrete points forming the collocation grid. Of major importance

* Received by the editors June 6, 1988; accepted for publication January 17, 1989.

† Unité de Mécanique Appliquée, Université Catholique de Louvain, B-1348 Louvain-La-Neuve, Belgium.

‡ Service de Métrologie Nucléaire, Université Libre de Bruxelles, B-1050 Bruxelles, Belgium, and Unité de Thermodynamique, Université Catholique de Louvain, B-1348 Louvain-La-Neuve, Belgium. The research of this author was supported by the Fonds National de la Recherche Scientifique (FNRS).

is therefore the choice of these collocation points. In this paper, the collocation points are zeros of orthogonal polynomials. The pseudospectral approximation is described generally in terms of Jacobi polynomials. The Chebyshev and Legendre approximations are two important particular cases that are easily deduced from the general theory. Most of the subsequent analysis will be performed for the Chebyshev method.

Preliminary computations of the condition number characterizing the pseudospectral matrix system reveals $O(N^4)$ conditioning, where $N+1$ is the number of degrees of freedom in a one-dimensional problem. This is a dismal performance in comparison with the $O(N^2)$ conditioning proposed by FD or FE techniques. Expecting to achieve better accuracy with a high-order approximation, we are faced with a loss of accuracy induced by the solution of the linear system. To overcome this difficulty, we plan to reduce the conditioning by preconditioning. Finite difference preconditioning was proposed by Orszag [21] and Morchoisne [20]. A theoretical analysis carried out by Haldenwang et al. [17] shows that FD preconditioning requires under-relaxation with an optimal value of the relaxation parameter being 4/7. Finite-element preconditioning appeared almost at the same time [5], [10], [11]. In Deville and Mund [10], the relaxation parameter is set to one and this value holds for a large class of problems. The behavior of the FE preconditioner seemed to be more robust and efficient than the FD preconditioner and this contrasting observation needed to get a sound explanation, which is given here through the spectrum analysis of the iteration operator.

In § 2, the pseudospectral approximation algorithm is presented for a general mixed boundary value problem of elliptic type. The basis functions of Lagrange interpolation are expressed through Jacobi polynomials. Chebyshev and Legendre cases are recovered for special values of the indices defining the general orthogonal polynomials. The matrix structure of the pseudospectral system is examined and some interesting conclusions may be drawn.

Section 3 analyses the finite-element preconditioning of pseudospectral approximations. Using functional notation, the preconditioned Richardson iterative procedure is introduced for the Dirichlet and Neumann conditions. The general case (Robin conditions) is then described. In § 4, numerical results are obtained. A complete analysis of the eigenvalue spectrum for the iteration operator is carried out for several mathematical problems including nonconstant coefficients. Linear, quadratic, and cubic Lagrangian elements and Hermite cubic elements are used as preconditioners. It is shown numerically that linear elements and Hermite elements provide the best results from the preconditioning point of view. Because of the larger bandwidth associated with Hermite elements, the final choice is clearly the linear element preconditioning. Afterward, several cases of two-dimensional problems are inspected within the framework of bilinear elements as preconditioners.

Section 5 is especially focused on the solution of an elliptic problem with corner singularities. As it is known from convergence analysis [4], the spectral rate of convergence is not attained. Nonetheless, it is revealed that the pseudospectral preconditioned algorithm performs extremely well. Such unexpected behavior is encouraging to promote pseudospectral calculations even for mathematical problems where geometric induced singularities decrease the convergence rates. The final section is devoted to conclusions.

2. The pseudospectral approximation algorithm.

This paper examines the numerical solution of two-dimensional elliptic boundary value problems with mixed boundary conditions:

$$Lu \triangleq -\bar{\nabla}(p(\bar{r})\bar{\nabla}u(\bar{r}))+q(\bar{r})u(\bar{r})=f(\bar{r}) \quad \forall \bar{r}\in\Omega, \tag{2.1a}$$

$$(2.1b) \qquad Bu \triangleq a(\bar{r})\left(p(\bar{r})\frac{\partial u}{\partial \mathbf{n}}\right) + b(\bar{r})u(\bar{r}) = g(\bar{r}) \quad \forall \bar{r} \in \partial\Omega.$$

The domain Ω is simply connected and bounded in the (x, y) plane; its boundary $\partial\Omega$ is piecewise smooth. The symbol $\partial/\partial\mathbf{n}$ denotes the normal derivative to $\partial\Omega$. We assume, moreover, conditions of uniform ellipticity to be satisfied [6]:

$$(2.2a) \qquad p(\bar{r}) \geqq \gamma > 0, \quad q(\bar{r}) \geqq 0 \quad \forall \bar{r} \in \Omega,$$

$$(2.2b) \qquad a(\bar{r}) + b(\bar{r}) > 0 \quad \forall \bar{r} \in \partial\Omega,$$

thus ensuring problem (2.1) is well posed. With properties (2.2), a unique solution of (2.1a), (2.1b) is known to exist in the Sobolev space $H^1(\Omega)$. For now, we restrict ourselves to problems having a C^∞ solution; later, we discuss the application of the algorithm to cases where corner singularities might hamper the convergence of spectral approximations.

Let (u, v) denote the scalar product of elements of $L^2(\Omega)$:

$$(2.3) \qquad (u, v) = \int_\Omega d\bar{r}\, u(\bar{r})v(\bar{r}).$$

In order to describe the pseudospectral approximation on the reference square $\Omega_I = [-1, 1]\otimes[-1, 1]$, we start by defining some notation. We set $N = (N_x, N_y) \in \mathcal{N} \times \mathcal{N}$ any couple of natural numbers (positive integers). Let $G_N^{(\alpha,\beta)}$ denote the tensor product of one-dimensional Gauss-Lobatto-Jacobi quadrature grids:

$$(2.4) \qquad G_N^{(\alpha,\beta)} = G_{x,N_x}^{(\alpha,\beta)} \otimes G_{y,N_y}^{(\alpha,\beta)},$$

where $G_{x,N_x}^{(\alpha,\beta)}$ and $G_{y,N_y}^{(\alpha,\beta)}$ are the ordered sets $\{z_j, j = 0, \cdots, N_z\}$ of the roots of

$$(2.5) \qquad (1 - z^2)\frac{d}{dz} P_{N_z}^{(\alpha,\beta)}(z) = 0,$$

in both space directions. The functions $P_{N_z}^{(\alpha,\beta)}(z)$ are the Jacobi polynomials of degree N_z and indices α and β. In particular, two important practical cases will be considered corresponding to Chebyshev polynomials of first kind ($\alpha = \beta = -\frac{1}{2}$) and Legendre polynomials ($\alpha = \beta = 0$). The pseudospectral approximation belongs to the family of weighted residual methods, whereby the residual is evaluated using an expansion of the dependent variable u_N into a finite sum of two-dimensional basis functions. These basis functions are obtained by tensor products over a set of one-dimensional polynomials $\{\phi_i\}$ and

$$(2.6) \qquad u_N = \sum_{i=0}^{N_x} \sum_{j=0}^{N_y} u_{ij}\phi_i(x)\phi_j(y).$$

Inserting (2.6) in (2.1), we obtain the residual $R[u_N]$:

$$(2.7) \qquad R[u_N] = \begin{cases} Lu_N(\bar{r}) - f(\bar{r}), & \forall \bar{r} \in \Omega, \\ Bu_N(\bar{r}) - g(\bar{r}), & \forall \bar{r} \in \partial\Omega. \end{cases}$$

We also introduce a set of test functions, $\{w_i(\bar{r}), i = 0, \cdots, N\}$, and require

$$(2.8) \qquad \int_\Omega d\bar{r}\, R[u_N]w_i(\bar{r}) = 0, \qquad i = 0, \cdots, N.$$

The pseudospectral method (equally known as the orthogonal collocation method) corresponds to the case where the w_i functions are Dirac distributions. Therefore, the partial differential equation (pde) is required to be solved *exactly* on a discrete set of interior nodes while the boundary conditions are enforced on $\partial\Omega$. This procedure depends very much on the choice of collocation nodes. Orthogonal collocation that is applied here rests on the nodes associated to the Gauss-Lobatto-Jacobi quadrature rule induced by (2.5) (see, for instance, [7] and [24]).

The most convenient functions ϕ_j correspond to the cardinal basis of the Lagrange interpolation problem on the grid $G_{z,N_z}^{(\alpha,\beta)}$. They are given by

$$\phi_j(z) = C_j \frac{(1-z^2)}{(z-z_j)} \frac{d}{dz} P_{N_z}^{(\alpha,\beta)}(z), \qquad z = x, y, \tag{2.9}$$

C_j being a normalization constant such that

$$\phi_j(z_k) = \delta_{jk}, \tag{2.10}$$

where δ denotes the Kronecker symbol. With a little algebra, we obtain

$$C_j = -\frac{1}{\bar{c}_j N_z(N_z+\alpha+\beta+1)} \frac{1}{P_{N_z}^{(\alpha,\beta)}(z_j)}, \tag{2.11}$$

with

$$\bar{c}_j = \begin{cases} 1/(\beta+1), & j=0, \\ 1, & j \in [1, N_z - 1], \\ 1/(\alpha+1), & j = N_z. \end{cases} \tag{2.12}$$

For the particular cases of Chebyshev and Legendre polynomials, the constant C_j is assigned to the classical values

$$C_j = \frac{(-1)^{j+1}}{(N_z^2 \bar{c}_j)}, \quad \bar{c}_0 = \bar{c}_{N_z} = 2, \quad \bar{c}_j = 1 \quad \forall j \in [1, N_z - 1] \tag{2.13a}$$

$$C_j = \frac{(-1)}{[N_z(N_z+1) q_{N_z}(z_j)]} \tag{2.13b}$$

where q_{N_z} represents the Legendre polynomial of degree N_z. The collocation abscissae for the Chebyshev case are given explicitly by

$$z_k = \cos\frac{\pi k}{N_z}, \qquad k \in [0, N_z], \tag{2.14}$$

while for other choices of Jacobi polynomials, they must be determined numerically (see [24]). The pseudospectral equations corresponding to problem (2.1a), (2.1b) may be written as follows:

$$\begin{aligned} &-\bar{\nabla}[p(\bar{r}_i)\bar{\nabla}u(\bar{r}_i)] + q(\bar{r}_i)u(\bar{r}_i) = f(\bar{r}_i) \quad \forall \bar{r}_i \in G_N^{(\alpha,\beta)} \cap \Omega, \\ &Bu(\bar{r}_i) = g(\bar{r}_i) \quad \forall \bar{r}_i \in G_N^{(\alpha,\beta)} \cap \partial\Omega. \end{aligned} \tag{2.15}$$

Equation (2.15) leads to the algebraic system of equations

$$L_{\mathrm{ps}}\bar{u} = \bar{f}, \qquad \bar{f} = [f(\bar{r}_i), g(\bar{r}_i)]^T, \tag{2.16}$$

where L_{ps} is the pseudospectral operator including the pde and the boundary conditions. The notation T denotes the transpose operation.

Some care must be taken with the boundary conditions, which deserve a few comments. In the sequel we shall assume that the boundary conditions are of the same type at all points of a side. Further, we shall denote by $\partial\Omega_D$ and $\partial\Omega_N$, respectively, the collection of sides where essential and natural boundary conditions apply to the solution of (2.1). Essential boundary conditions correspond to the situation where $a(\bar{r})=0$ in (2.1.b). At vertices joining $\partial\Omega_D$ and $\partial\Omega_N$ in the pseudospectral equation (2.15), the Dirichlet boundary condition will systematically be preferred, whereas at vertices joining natural condition sides, a linear combination of the two conditions will be the rule. This strategy has proven to be the most efficient (cf. [3]).

Notice that (2.15) involves first- and second-order derivatives. We need to express these derivatives in terms of the basis functions $\phi_j(z)$ given by (2.9). For the sake of simplicity, only one-dimensional expressions will be given, the extension to higher spatial dimensions being trivial because of the tensor-product character of the approximation. In the one-dimensional case, (2.6) reduces to the following:

$$u_N(z)=\sum_{j=0}^{N_z} u_j\phi_j(z), \tag{2.17}$$

and therefore

$$\frac{d^i u_N(z)}{dz^i}=\sum_{j=0}^{N_z} u_j\frac{d^i\phi_j(z)}{dz^i}. \tag{2.18}$$

In order to evaluate the derivatives in the right-hand side of (2.18), one observes that Jacobi polynomials satisfy the second-order differential equation

(2.19)
$$(1-z^2)P_{N_z}^{(\alpha,\beta)''}(z)+(\beta-\alpha-z(\alpha+\beta+2))P_{N_z}^{(\alpha,\beta)'}(z)+N_z(N_z+\alpha+\beta+1)P_{N_z}^{(\alpha,\beta)}(z)=0.$$

With the help of (2.19) one obtains, after some algebra,

$$\left.\frac{d}{dz}\phi_j(z)\right|_{z=z_k}=\frac{C_k}{C_j}\frac{1}{z_k-z_j}\frac{P_{N_z}^{(\alpha,\beta)}(z_k)}{P_{N_z}^{(\alpha,\beta)}(z_j)},\qquad z_k\neq z_j,\ j,k\in[0,N_z] \tag{2.20a}$$

$$\left.\frac{d}{dz}\phi_j(z)\right|_{z=z_j}=-\frac{1}{2}\frac{\beta-\alpha-(\alpha+\beta)z_j}{1-z_j^2},\qquad j\in[1,N_z-1], \tag{2.20b}$$

$$\left.\frac{d}{dz}\phi_0(z)\right|_{z=-1}=-\frac{1}{2}\frac{N_z(N_z+\alpha+\beta+1)-\alpha}{\beta+2}, \tag{2.20c}$$

$$\left.\frac{d}{dz}\phi_{N_z}(z)\right|_{z=1}=\frac{1}{2}\frac{N_z(N_z+\alpha+\beta+1)-\beta}{\alpha+2}. \tag{2.20d}$$

The special cases of Chebyshev and Legendre polynomials yield the classical expressions that can be found in the literature (see [14]). The constants C_j in (2.20) are those defined by (2.11)-(2.12).

For the second-order derivatives, we have

$$\left.\frac{d^2}{dz^2}\phi_j(z)\right|_{z=z_k}=-\frac{1}{C_j}\frac{((\beta-\alpha-(\alpha+\beta)z_k)(z_k-z_j)+2(1-z_k^2))}{(z_k-z_j)^2(1-z_k^2)}\frac{P_{N_z}^{(\alpha,\beta)}(z_k)}{P_{N_z}^{(\alpha,\beta)}(z_j)}, \tag{2.21a}$$

$$z_k\neq z_j,\qquad j\in[0,N_z],\qquad k\in[1,N_z-1],$$

$$\left.\frac{d^2}{dz^2}\phi_j(z)\right|_{z=-1}=-\frac{1}{2}\frac{C_0}{C_j}\left(\frac{4}{(1+z_j)^2}-\frac{2}{\beta+2}\frac{N_z(N_z+\alpha+\beta+1)-\alpha}{1+z_j}\right)\frac{P_{N_z}^{(\alpha,\beta)}(-1)}{P_{N_z}^{(\alpha,\beta)}(z_j)}, \tag{2.21b}$$

$$j\in[1,N_z],$$

$$\text{(2.21c)} \quad \left.\frac{d^2}{dz^2}\phi_j(z)\right|_{z=1} = -\frac{1}{2}\frac{C_{N_z}}{C_j}\left(\frac{4}{(1-z_j)^2} - \frac{2}{\alpha+2}\frac{N_z(N_z+\alpha+\beta+1)-\beta}{1-z_j}\right)\frac{P_{N_z}^{(\alpha,\beta)}(1)}{P_{N_z}^{(\alpha,\beta)}(z_j)}, \quad j\in[0, N_z-1],$$

$$\text{(2.21d)} \quad \left.\frac{d^2}{dz^2}\phi_j(z)\right|_{z=z_j} = -\frac{1}{3(1-z_j^2)}\Bigg(N_z(N_z+\alpha+\beta+1)-2(\alpha+\beta) - \frac{(\beta-\alpha-(\alpha+\beta)z_j)(\beta-\alpha-(\alpha+\beta+4)z_j)}{1-z_j^2}\Bigg), \quad j\in[1, N_z-1],$$

$$\text{(2.21e)} \quad \left.\frac{d^2}{dz^2}\phi_0(z)\right|_{z=-1} = \frac{1}{4}\frac{(N_z(N_z+\alpha+\beta+1)-2\alpha)(N_z(N_z+\alpha+\beta+1)-(\alpha+\beta+2))}{(\beta+2)(\beta+3)},$$

$$\text{(2.21f)} \quad \left.\frac{d^2}{dz^2}\phi_{N_z}(z)\right|_{z=1} = \frac{1}{4}\frac{(N_z(N_z+\alpha+\beta+1)-2\beta)(N_z(N_z+\alpha+\beta+1)-(\alpha+\beta+2))}{(\alpha+2)(\alpha+3)}.$$

Again (2.21) provide the user with well-known expressions for the special cases related to Chebyshev and Legendre approximations.

Let us now examine the structure of the L_{ps} matrix and some associated operation counts. For that purpose we consider the matrix corresponding to the Laplace equation with Dirichlet boundary conditions and $N_x = N_y = N$. Figure 1 displays the topological structure of the pseudospectral system for $N = 7$ with unknowns u_{ij} numbered in the

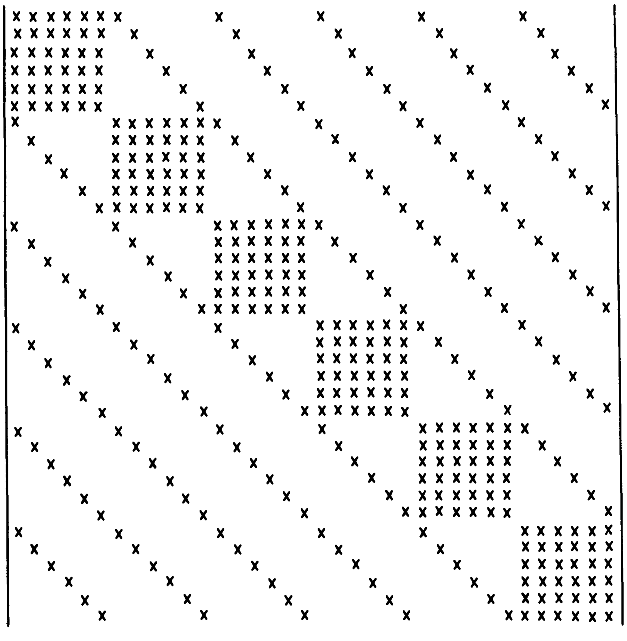


FIG. 1. *Topological structure of the pseudospectral system L_{ps} for the two-dimensional Laplace equation with homogeneous Dirichlet conditions and $N_x = N_y = 7$. The unknowns are numbered in the lexicographic order.*

classical lexicographic order. The matrix has a very peculiar block structure. Each diagonal block is related to a row of unknowns, while off-diagonal blocks are diagonal and related to unknowns in the vertical direction. Contrary to a widespread opinion, the pseudospectral matrix is not full. But long-range coupling between unknowns gives an exceptionally large bandwidth (B), almost equal to the total number of unknowns: $B_{\text{ps}} = N^2 - N$. Obviously, other ordering schemes might be chosen such as, for instance, the classical antidiagonal scheme. The resulting bandwidth is slightly reduced but still large as shown in Fig. 2. By contrast, finite-difference and finite-element methods have quite smaller bandwidths since, typically $B_{\text{fe}} = N+1$. Figure 3 shows the topological structure of the bilinear Lagrangian FE stiffness matrix associated to the same space grid and with lexicographic ordering of the unknowns.

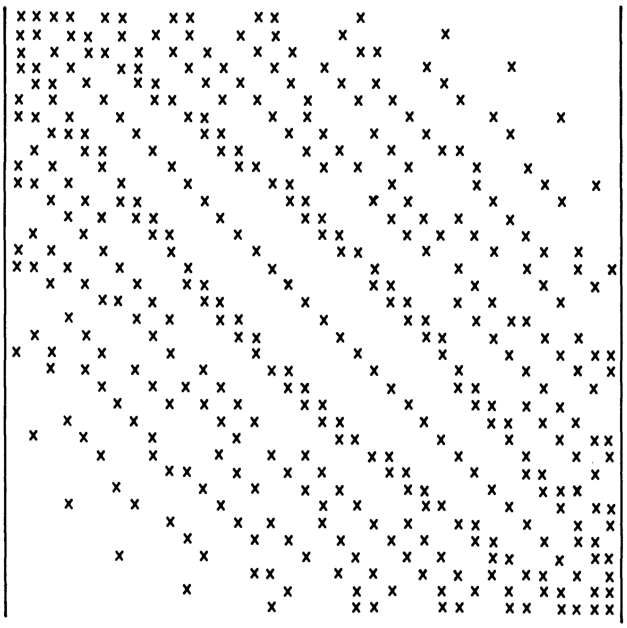


FIG. 2. *Topological structure of the pseudospectral system L_{ps} for the same problem and the same mesh grid as in Fig. 1. The unknowns are numbered in the antidiagonal order.*

If we use a direct inversion technique to solve the pseudospectral algebraic system (2.16), Golub and Van Loan's estimates of flops (floating point operations per second) for LU factorization (w) and forward and backward solve (r) give (see [12])

$$w_{\text{ps}}(N) = (N^6 - N^3)/3, \qquad r_{\text{ps}}(N) = N^4 \tag{2.22}$$

whereas for bilinear finite elements on the same grid, we obtain $w_{\text{fe}} = N^4 + O(N^3)$ and $r_{\text{fe}} = 2N^3 + O(N^2)$. It is interesting to note that, because of the reduced bandwidth, the factorization of this finite-element matrix is only slightly more expensive than the forward and backward solve of the factorized pseudospectral system. We shall come back to this point later.

Little is known about the properties of the pseudospectral matrix L_{ps}. It is not symmetric. However, because of the symmetry of the Chebyshev (or Legendre) abscissae, the matrix coefficients of L_{ps} satisfy

$$(L_{\text{ps}})_{i,j} = (L_{\text{ps}})_{(N_z-1)^2+1-i,(N_z-1)^2+1-j}, \qquad i,j = 1, \cdots, \left[\frac{(N_z-1)^2}{2}\right] \tag{2.23}$$

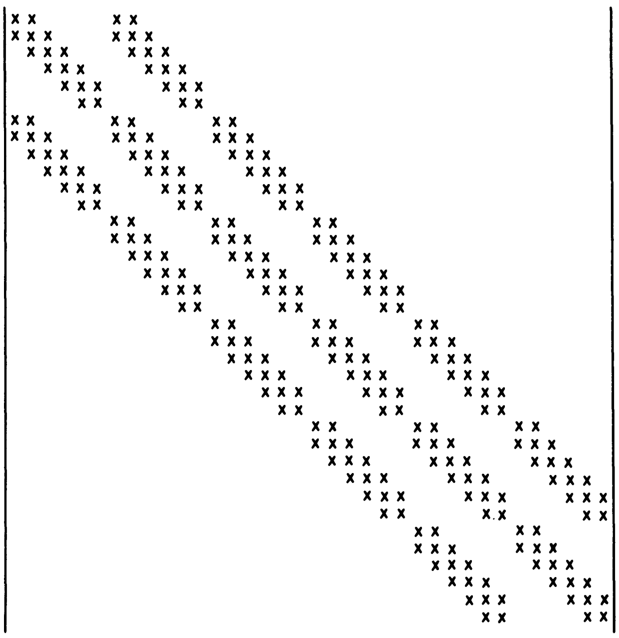


FIG. 3. *Topological structure of the bilinear* FE *stiffness matrix for the same problem, the same mesh grid and same unknowns ordering as in Fig.* 1.

for problems where the diffusion coefficient $p(\bar{r})$ has the symmetry properties of the domain Ω. Gottlieb and Lustman have shown that the collocation operator of the heat equation has negative real eigenvalues (see [13]). Their proof rests on the use of Hurwitz polynomials. There are good prospects that their result might be extended to other collocation grids.

One of the major drawbacks of pseudospectral calculations lies in the value of the condition number $\kappa(L_{\mathrm{ps}}) = O((N^4)^d)$, which makes large systems increasingly ill-conditioned. In the previous estimate of κ, d is the number of space dimensions. This, once again, is in sharp contrast with FD or FE methods that have $\kappa = O((N^2)^d)$. Therefore, instead of solving (2.16), L_{ps} is split into two parts:

$$L_{\mathrm{ps}} = \hat{L} + (L_{\mathrm{ps}} - \hat{L}), \tag{2.24}$$

where $\hat{L}$ is a nonsingular approximation of L_{ps}. Introducing (2.24) into (2.16), we obtain

$$\hat{L}\bar{u} = \hat{L}\bar{u} - (L_{\mathrm{ps}}\bar{u} - \bar{f}), \tag{2.25}$$

or, more simply,

$$\bar{u} = \bar{u} - \hat{L}^{-1}(L_{\mathrm{ps}}\bar{u} - \bar{f}). \tag{2.26}$$

This result is the heart of the preconditioned Richardson iteration procedure, which is written

$$\bar{u}^{k+1} = \bar{u}^k - \alpha_k \hat{L}^{-1}(L_{\mathrm{ps}}\bar{u}^k - \bar{f}) \tag{2.27}$$

where k denotes an iteration index. If the sequence of iterates converges, (2.27) yields the solution of (2.16). The parameter α_k controls the rate of convergence of the iterative process. Its numerical value (that might change from iteration to iteration) will depend on the spectral radius $\rho(I - \hat{L}^{-1}L_{\mathrm{ps}})$ of the iteration operator. Two choices of the splitting matrix $\hat{L}$ seem quite natural. The first relies on finite differences (see [5], [17], [20], [21]), and the second on finite elements (see [5], [10], [11]).

A theoretical study for the FD preconditioning of the second derivative operator [17] shows that the eigenvalue spectrum of $\hat{L}^{-1}L_{\text{ps}}$ lies in the interval $(1, \pi^2/4)$. The optimal value of the relaxation factor is

$$\alpha_{\text{opt}} = \frac{2}{\lambda_{\min} + \lambda_{\max}}, \tag{2.28}$$

where $\lambda_{\min}$ and $\lambda_{\max}$ are the minimum and maximum eigenvalues of $\hat{L}^{-1}L_{\text{ps}}$ [15]. In the FD preconditioning case, $\alpha_{\text{opt}} \cong 4/7$. This value differs very much from the relaxation factor used by Deville and Mund in the FE preconditioning case [10] that was systematically set to one for broad classes of differential operators. The justification of this choice is the central subject of the present work.

3. The finite-element preconditioning of pseudospectral approximations. As this paper is aimed at finite-element preconditioning, we first introduce some specific notation referring to low-order polynomial representations. In the sequel, we closely follow and extend the approach of Canuto and Pietra (see [3]).

Let $\mathcal{R}_N$ represent the collection of rectangles $\{R_i, i = 1, \cdots, N_xN_y\}$ whose vertices are four neighboring gridpoints of $G_N^{(\alpha,\beta)}$ (i.e., $\mathcal{R}_N = \bigcup_i R_i$). Also let Q_n denote the space of all polynomials in x, y of degree less than or equal to n with respect to each variable. We shall represent by $V_{n,h}^L$ and $V_{3,h}^H$, the Lagrange finite-element spaces of degree n and the bicubic Hermite space on the "triangulation" $\mathcal{R}_N$:

$$V_{n,h}^L = \{v_h \in C^0(\bar{\Omega}) \mid \forall R_i \in \mathcal{R}_N,\ v_{h|R_i} \in Q_n\}, \tag{3.1a}$$

$$V_{3,h}^H = \{v_h \in C^1(\bar{\Omega}) \mid \forall R_i \in \mathcal{R}_N,\ v_{h|R_i} \in Q_3\}. \tag{3.1b}$$

These spaces are known in the literature to be made of two-rectangles of type (n) and Bogner-Fox-Schmit rectangles, respectively, (see [6]). For practical purposes, the value of n in (3.1a) will be limited to three (bicubic Lagrange elements at most). Further, let $\mathcal{G}_{n,N}^L$ and $\mathcal{G}_{3,N}^H$ represent the set of finite-element nodes for the (N_x, N_y) partition of $\bar{\Omega}$ in the Lagrange and Hermite cases.

We denote by $l(\mathcal{G}_{n,N}^L)$ and $l(\mathcal{G}_{3,N}^H)$, the sets of linear functionals corresponding to the degrees of freedom of $V_{n,h}^L$ and $V_{3,h}^H$. In the particular case of *one* finite element covering the unit square, these sets are such that

$$l(\mathcal{G}_{n,N}^L)(u) = \left(u\left(\frac{i}{n}, \frac{j}{n}\right);\quad i, j = 0, \cdots, n\right), \tag{3.2a}$$

$$l(\mathcal{G}_{3,N}^H)(u) = \left(\frac{\partial^\sigma}{\partial x^{\sigma_1}\partial y^{\sigma_2}} u \bigg|_{\substack{x=0,1\\ y=0,1}},\ \sigma = \sigma_1 + \sigma_2,\quad \sigma_1, \sigma_2 = 0, 1\right), \tag{3.2b}$$

whereas in the general case, similar relations apply for each rectangle R_i, after mapping onto the simplex $[0, 1]\otimes[0, 1]$. Later, we shall refer to an element (or a subset) of the set of linear functionals at a given node of the finite-element grid by using $l(\mathcal{G}_{n,N}^L)(\cdot)|_{\bar{r}_k}$ or $l(\mathcal{G}_{3,N}^H)(\cdot)|_{\bar{r}_k}$.

The finite-element grids $\mathcal{G}_{n,N}^L$ and $\mathcal{G}_{3,N}^H$ satisfy the relations

$$G_N^{(\alpha,\beta)} \subseteq \mathcal{G}_{n,N}^L, \qquad G_N^{(\alpha,\beta)} \equiv \mathcal{G}_{3,N}^H. \tag{3.3}$$

Given a smooth function g on $\partial\Omega_D$, let $V_{n,h}^L(g)$ denote the affine space of functions of $V_{n,h}^L$ that interpolate g at the essential boundary nodes of $\mathcal{G}_{n,N}^L$:

$$V_{n,h}^L(g) = (v_h \in V_{n,h}^L \mid l(\mathcal{G}_{n,N}^L)(v_h - g)|_{\bar{r}_k} = 0,\ \forall \bar{r}_k \in \mathcal{G}_{n,N}^L \cap \partial\Omega_D). \tag{3.4a}$$

Similarly, in the Hermite case, we introduce $V^H_{3,h}(g)$ such that

$$V^H_{3,h}(g) = (v_h \in V^H_{3,h} | l(\mathscr{G}^H_{3,N})(v_h - g)|_{\bar{r}_k} = 0, \forall \bar{r}_k \in \mathscr{G}^H_{3,N} \cap \partial\Omega_D). \tag{3.4b}$$

In the particular situation where g is identically equal to zero, we shall write

$$V^{L,0}_{n,h} \equiv V^L_{n,h}(0), \qquad V^{H,0}_{3,h} \equiv V^H_{3,h}(0). \tag{3.5}$$

The preconditioning algorithm uses three families of interpolation operators, I, $\mathscr{I}$, and J, that are needed to go from spectral to finite-element representations back and forth. Both the I and $\mathscr{I}$ families are made of two-dimensional operators, whereas the operators J are one dimensional.

Let $I^L_{n,h}$ and $I^H_{3,h}$ be the Lagrange and Hermite finite-element interpolation operator on the grids $\mathscr{G}^L_{n,N}$ and $\mathscr{G}^H_{3,N}$. These operators are defined by

$$I^L_{n,h}:$$

$$\begin{aligned} &\forall u(\bar{r}) \in C^0(\bar{\Omega}) \to I^L_{n,h}u = v_h(\bar{r}) \in V^L_{n,h}, \\ &l(\mathscr{G}^L_{n,N})(I^L_{n,h}u - u)|_{\bar{r}_k} = 0 \quad \forall \bar{r}_k \in \mathscr{G}^L_{n,N}, \end{aligned} \tag{3.6}$$

and

$$I^H_{3,h}:$$

$$\begin{aligned} &\forall u(\bar{r}) \in C^1(\bar{\Omega}) \to I^H_{3,h}u = v_h(\bar{r}) \in V^H_{3,h}, \\ &l(\mathscr{G}^H_{3,N})(I^H_{3,h}u - u)|_{\bar{r}_k} = 0, \quad \forall \bar{r}_k \in \mathscr{G}^H_{3,N}. \end{aligned} \tag{3.7}$$

In the same way, we represent by $I^{L,0}_{n,h}$ and $I^{H,0}_{3,h}$, finite-element interpolation operators setting to zero the values of the linear functionals on the boundary nodes, i.e.:

$$I^{L,0}_{n,h}:$$

$$\begin{aligned} &\forall u(\bar{r}) \in C^0(\bar{\Omega}) \to I^{L,0}_{n,h}u = v_h(\bar{r}) \in V^{L,0}_{n,h}, \\ &l(\mathscr{G}^L_{n,N})(I^{L,0}_{n,h}u - u)|_{\bar{r}_k} = 0, \quad \forall \bar{r}_k \in \mathscr{G}^L_{n,N} \cap \Omega, \\ &l(\mathscr{G}^L_{n,N})(I^{L,0}_{n,h}u)|_{\bar{r}_k} = 0, \quad \forall \bar{r}_k \in \mathscr{G}^L_{n,N} \cap \partial\Omega, \end{aligned} \tag{3.8}$$

and

$$I^{H,0}_{3,h}:$$

$$\begin{aligned} &\forall u(\bar{r}) \in C^1(\bar{\Omega}) \to I^{H,0}_{3,h}u = v_h(\bar{r}) \in V^{H,0}_{3,h}, \\ &l(\mathscr{G}^H_{3,N})(I^{H,0}_{3,h}u - u)|_{\bar{r}_k} = 0, \quad \forall \bar{r}_k \in \mathscr{G}^H_{3,N} \cap \Omega, \\ &l(\mathscr{G}^H_{3,N})(I^{H,0}_{3,h}u)|_{\bar{r}_k} = 0 \quad \forall \bar{r}_k \in \mathscr{G}^H_{3,N} \cap \partial\Omega. \end{aligned} \tag{3.9}$$

Basically, these operators will be used to project spectral residuals of the pde into the finite-element spaces $V^{L,0}_{n,h}$ or $V^{H,0}_{3,h}$.

We denote by $\mathscr{I}_N$, the two-dimensional spectral interpolation operator acting on a set of discrete values to produce an element of $\mathscr{P}^{(\alpha,\beta)}_N$, the space of functions obtained by tensor product of Jacobi polynomials in both space directions. The operator $\mathscr{I}_N$ is the same, regardless of the finite-element choice. We have the following

$$\mathscr{I}_N:$$

$$\begin{aligned} &\forall v_h(\bar{r}) \in V^L_{n,h} \to \mathscr{I}_N v_h = u(\bar{r}) \in \mathscr{P}^{(\alpha,\beta)}_N, \\ &l(\mathscr{G}^L_{n,N})(\mathscr{I}_N v_h - v_h)|_{\bar{r}_k} = 0, \quad \forall \bar{r}_k \in G^{(\alpha,\beta)}_N, \end{aligned} \tag{3.10}$$

for Lagrange finite elements and also

$$\begin{aligned}&\mathcal{J}_N:\\ &\forall v_h(\bar r)\in V^H_{3,h}\to \mathcal{J}_N v_h = u(\bar r)\in \mathcal{P}^{(\alpha,\beta)}_N,\\ &l(\mathcal{G}^H_{3,N})(\mathcal{J}_N v_h - v_h)|_{\bar r_k}=0, \qquad \bar r_k\in G^{\alpha,\beta}_N \text{ and } \sigma=0,\end{aligned}\tag{3.11}$$

for Hermite elements. Interpolation occurs at the collocation nodes only. Both the "internal" degrees of freedom of Lagrange elements with $n>1$ and the nodal derivatives of Hermite elements are therefore discarded. $\mathcal{J}_N$ is merely a convenient tool to identify an element of $\mathcal{P}^{(\alpha,\beta)}_N$, using its nodal values.

The one-dimensional interpolation operator J is used for the treatment of the boundary terms arising in the variational formulation of Neumann problems. We postpone its definition until § 3.2.

3.1. Preconditioning of Dirichlet problems. We start by investigating the finite-element preconditioning of homogeneous Dirichlet boundary value problems (i.e., $a(\bar r)=0, g(\bar r)=0$, and $b(\bar r)=1$ in (2.1b)). The algorithm will be outlined with the Lagragian finite elements of degree n. The case of the Hermite elements is easily transposable.

Given a function $F\in L^2(\Omega)$, we denote by $w_n=\hat L^{-1}_{\text{fe}}[F]$ the finite-element solution of a problem that, in the standard Galerkin approach, is written

$$\begin{aligned}&w_h\in V^{L,0}_{n,h}\\ &\int_\Omega d\bar r\,(p\bar\nabla w_h\bar\nabla v_h+qw_hv_h)=\int_\Omega d\bar r\,Fv_h \quad \forall v_h\in V^{L,0}_{n,h}.\end{aligned}\tag{3.12}$$

Actually, w_h results from the inversion of an algebraic system:

$$(S^L_{n,N})\cdot\bar w=(M^L_{n,N})\cdot\bar F,\tag{3.13}$$

where $(S^L_{n,N})$ and $(M^L_{n,N})$ are the stiffness and mass matrices associated to the problem and its discretization.

A genuine FE discretization results only when the numerical quadratures appearing in (3.12) are made exactly (at least *after* projection of the coefficient functions $p(\bar r)$ and $q(\bar r)$ into $V^L_{n,h}$). If, instead, we use approximate quadrature rules (i.e., trapezoidal, Simpson, etc.) the algebraic system (3.13) reduces to a FD scheme. The main difference between the two cases lies in the mass matrix whose presence in the FE discretization is crucial and explains its remarkable preconditioning properties. We shall use this in order to compare the spectral properties of both preconditioning techniques in the same software environment.

The preconditioned Richardson iterations are implemented as follows. Let w^0_h be the solution of

$$\begin{aligned}&w^0_h\in V^{L,0}_{n,h}\\ &\int_\Omega d\bar r(p\bar\nabla w^0_h\bar\nabla v_h+qw^0_hv_h)=\int_\Omega d\bar r f v_h \quad \forall v_h\in V^{L,0}_{n,h}\end{aligned}\tag{3.14}$$

and let

$$u^0=\mathcal{J}_N w^0_h\tag{3.15}$$

represent the projection of w^0_h into $\mathcal{P}^{(\alpha,\beta)}_N$.

Using the interpolation operators defined earlier, the subsequent iterations may be written, in obvious notation:

$$u^{k+1}=u^k-\alpha_k \mathcal{J}_N \hat{L}_{\text{fe}}^{-1}(I_{n,h}^{L,0}(L_{\text{ps}}u^k-f)), \qquad k=0,1,\cdots. \tag{3.16}$$

The iterative process is carried out until convergence that, in practice, is controlled by the residual $(L_{\text{ps}}u^k-f)$.

The evaluation of the residual is an important part of the algorithm. For any values of the Jacobi polynomial indices (α,β), the pseudospectral matrix L_{ps} is easily set up, using the relations (2.9)-(2.10) and (2.20)-(2.21). Evaluation of the residual entails N^4 flops, which is approximately the numerical work corresponding to the factorization of the FE stiffness matrix. If, however, a Gauss-Lobatto-Chebyshev grid is selected $(\alpha=\beta=-\frac{1}{2})$, evaluation of the residual can be made straightforwardly using direct and inverse FFT (see [9]). In that case, the numerical work is reduced to $2N^2\log_2 2N$, which makes it cheaper than even the forward and backward solve parts of the FE preconditioning.

The convergence of the numerical scheme (3.14)-(3.16) is governed by the spectral radius of the iteration operator $(I-\hat{L}_{\text{fe}}^{-1}\cdot I_{n,h}^{L,0}\cdot L_{\text{ps}})$. Using the stiffness and mass matrices defined in (3.13), the iteration operator may be written, equivalently:

$$A\triangleq I-(S_{n,N}^L)^{-1}\cdot(M_{n,N}^L)\cdot(I_{n,h}^{L,0})\cdot(L_{\text{ps}}), \tag{3.17}$$

where the symbols $(I_{n,h}^{L,0})$ and (L_{ps}) represent the matrix form of the (abstract) operators $I_{n,h}^{L,0}$ and L_{ps}. The coefficients of $(I_{n,h}^{L,0})$ are easily calculated using the cardinal basis (2.9) and its derivatives (2.20) and (2.21). Note that in the simple case of linear FE preconditioning, $(I_{n,h}^{L,0})$ reduces to the identity matrix.

A necessary and sufficient condition for convergence of (3.16) is that $\rho(A)<1$. A rough estimate of the number of iterations (n) required to reduce the error norm by a factor ζ is then given by

$$n=-\frac{\log\zeta}{R_\infty(A)}, \tag{3.18}$$

where $R_\infty(A)=-\log\rho(A)$ is the asymptotic rate of convergence of the iteration matrix (see [15]).

3.2. Preconditioning of Neumann problems. Let us turn to the finite-element preconditioning of a homogeneous Neumann problem (i.e., $b(\bar{r})$, $g(\bar{r})=0$, and $a(\bar{r})=1$ in (2.1b); $\partial\Omega\equiv\partial\Omega_N$).

The treatment of boundary terms in the weak formulation of a Neumann problem requires a one-dimensional interpolation operator that has been termed J in the introduction of this section. Depending on the choice of the FE discretization of the problem (Lagrange or Hermite), the operator will be called $J_{n,h}^L$ or $J_{3,h}^H$.

We shall further denote by $T_{n,h}^L(\partial\Omega)$ and $T_{3,h}^H(\partial\Omega)$, the FE trace spaces associated to $V_{n,h}^L$ and $V_{3,h}^H$, respectively. The definitions of $J_{n,h}^L$ and $J_{3,h}^H$ are the following:

$$\begin{gathered} J_{n,h}^L: \\ \forall u(\bar{r})\in L^2(\partial\Omega)\rightarrow J_{n,h}^L u=t_h(\bar{r})\in T_{n,h}^L(\partial\Omega), \\ l(\mathcal{G}_{n,N}^L)(J_{n,h}^L u-u)|_{\bar{r}_k}=0 \quad \forall \bar{r}_k\in\mathcal{G}_{n,N}^L\cap\partial\Omega, \end{gathered} \tag{3.19}$$

and

$$\begin{gathered} J_{3,h}^H: \\ \forall u(\bar{r})\in L^2(\partial\Omega)\rightarrow J_{3,h}^H u=t_h(\bar{r})\in T_{3,h}^H(\partial\Omega), \\ l(\mathcal{G}_{3,N}^H)(J_{3,h}^H u-u)|_{\bar{r}_k}=0 \quad \forall \bar{r}_k\in\mathcal{G}_{3,N}^H\cap\partial\Omega. \end{gathered} \tag{3.20}$$

Special notice must be given for the one-dimensional Lagrange interpolation operators $J^L_{n,h}$, which will be made clear after outline of the iterative procedure. Once again, the developments will use the Lagrangian finite element of degree n. Transposition to the Hermite element may be easily carried out.

Given a function $F \in L^2(\Omega)$, and a function $\psi \in L^2(\partial\Omega)$ we represent by $w_h = \hat{L}_{\text{fe}}^{-1}[F, \psi]$, the FE solution of the following problem:

$$(3.21)\qquad \begin{aligned} & w_h \in V^L_{n,h}, \\ & \int_\Omega d\bar{r}\,(p\bar{\nabla} w_h \bar{\nabla} v_h + q w_h v_h) = \int_\Omega d\bar{r}\, F v_h + \int_{\partial\Omega} d\bar{r}\, \psi v_h \quad \forall v_h \in V^L_{n,h}. \end{aligned}$$

The iterative algorithm for the FE preconditioning of the homogeneous Neumann problem (2.1) proceeds as follows. Let w_h^0 be the solution of

$$(3.22)\qquad \begin{aligned} & w_h^0 \in V^L_{n,h}, \\ & \int_\Omega d\bar{r}\,(p\bar{\nabla} w_h^0 \bar{\nabla} v_h + q w_h^0 v_h) = \int_\Omega d\bar{r}\, f v_h \quad \forall v_h \in V^L_{n,h}. \end{aligned}$$

Projection of this function into $\mathcal{P}_N^{(\alpha,\beta)}$ with $\mathcal{J}_N$ given by (3.10), yields the initial approximation

$$(3.23)\qquad u^0 = \mathcal{J}_N w_h^0.$$

Subsequent iterations are performed according to the scheme

$$(3.24)\qquad u^{k+1} = u^k - \alpha_k \mathcal{J}_N \hat{L}_{\text{fe}}^{-1}\left(I^{L,0}_{n,h}(L_{\text{ps}} u^k - f), J^L_{n,h}\left(p\frac{\partial u^k}{\partial \mathbf{n}}\right)\right), \qquad k = 0, 1, \cdots.$$

The difference between (3.24) and (3.16) lies mainly in the presence of a boundary residual originating from the natural conditions.

If the sequence $\{u^k \mid k \in \mathcal{N}\}$ generated by (3.22)-(3.24) converges to a limit $u^\infty \in \mathcal{P}_N^{\alpha,\beta}$ and the sequence $\{\alpha_k \mid k \in \mathcal{N}\}$ is bounded away from zero, then we have

$$(3.25)\qquad \hat{L}_{\text{fe}}^{-1}\left(I^{L,0}_{n,h}(L_{\text{ps}} u^\infty - f), J^L_{n,h}\left(p\frac{\partial u^\infty}{\partial \mathbf{n}}\right)\right) = 0,$$

or also, because of (3.21)

$$(3.26)\qquad \int_\Omega d\bar{r}\, I^{L,0}_{n,h}(L_{\text{ps}} u^\infty - f) v_h + \int_{\partial\Omega} d\bar{r}\, J^L_{n,h}\left(p\frac{\partial u^\infty}{\partial \mathbf{n}}\right) v_h = 0 \quad \forall v_h \in V^L_{n,h}.$$

In particular, for $v_h = I^{L,0}_{n,h}(L_{\text{ps}} u^\infty - f)$, test function whose value vanishes identically along $\partial\Omega$ (see (3.8)), we conclude that

$$(3.27)\qquad I^{L,0}_{n,h}[L_{\text{ps}} u^\infty - f] \equiv 0,$$

which constrains u^∞ to satisfy the collocation equations.

Insertion of (3.27) into (3.26) further yields

$$(3.28)\qquad \int_{\partial\Omega} d\bar{r}\, J^L_{n,h}\left(p\frac{\partial u^\infty}{\partial \mathbf{n}}\right) t_h = 0 \quad \forall t_h \in T^L_{n,h}.$$

Unfortunately, the argument developed above for the residual of the equation is no longer applicable. This is due to the behavior of $(\partial u^{\infty}/\partial \mathbf{n})$ at a corner. If we consider the limit values of the normal derivatives on two consecutive faces of $\partial\Omega$ as we approach the corner point, most of the time $(\partial u^{\infty}/\partial \mathbf{n})$ will be discontinuous, whereas the function t_h must be continuous. Therefore, we cannot enforce on the boundary residual a constraint similar to (3.27). All we can expect is that, because of its orthogonality properties (3.28) with respect to the test functions of $T^L_{n,h}$, $J^L_{n,h}$ $(p\,\partial u^{\infty}/\partial \mathbf{n})$ is globally small. In fact, this is what happens in practice as will be shown later. It is possible however, as shown by Canuto and Pietra [3], to enforce the collocation constraint exactly. We have therefore to modify slightly the interpolation operator $J^L_{n,h}$ at the corner nodes, into a linear combination of the normal derivatives at the adjoining sides.

Let us emphasize that these remarks do not apply to the preconditioning with cubic Hermite polynomials since, in that case, the boundary conditions may be satisfied exactly along $\partial\Omega$.

3.3. The general case. Having treated separately the Dirichlet and Neumann problems, we are now in the position to describe the FE preconditioning of the general boundary value problem (2.1). It is only a matter of choosing the right projection spaces.

Let $g(\bar{r})$ be decomposed into $g_D(\bar{r}) \cup g_N(\bar{r})$, where $g_D(\bar{r})$ and $g_N(\bar{r})$ are the values of $g(\bar{r})$ on the "essential" and "natural" parts of the boundary. As previously, we denote by $w_h = \hat{L}_{\text{fe}}^{-1}[F, \psi]$ the FE solution of (3.21).

The iterative algorithm for the FE preconditioning of (2.1) runs as follows. We first determine w_h^0 such that

$$\begin{aligned} &w_h^0 \in V^L_{n,h}(g_D), \\ &\int_\Omega d\bar{r}(p\bar{\nabla} w_h^0 \bar{\nabla} v_h + q w_h^0 v_h) = \int_\Omega d\bar{r} f v_h + \int_{\partial\Omega} d\bar{r}\frac{1}{a}(g_N - b w_h^0) v_h \quad \forall v_h \in V^{L,0}_{n,h}. \end{aligned} \tag{3.29}$$

Then, after projection into $\mathcal{P}_N^{(\alpha,\beta)}$

$$u^0 = \mathcal{J}_N w_h^0,$$

successive approximations of the pseudospectral solution are given by

$$u^{k+1} = u^k - \alpha_k \mathcal{J}_N \hat{L}_{\text{fe}}^{-1}\left(I^{L,0}_{n,h}(L_{\text{ps}} u^k - f), J^L_{n,h}\left(ap\frac{\partial u^k}{\partial \mathbf{n}} + bu^k - g_N \right)\right), \quad k = 0, 1, \cdots. \tag{3.30}$$

The conclusions of § 3.2 can readily be transposed to (3.30).

4. Numerical results. In this section we present a series of numerical results for various one- and two-dimensional Dirichlet and Neumann boundary value problems. We shall first describe the eigenvalue properties of the iteration operator (3.17). Then, we shall verify the effectiveness of the preconditioning algorithm by looking at its convergence speed.

Table 1 displays the lower and upper bounds $(|\lambda_m|, |\lambda_M|)$ of the eigenvalue spectrum and the condition number $\kappa(\hat{L}_{\text{fe}}^{-1} L_{\text{ps}})$ for three *ordinary* differential operators with Dirichlet boundary conditions on $(-1, +1)$. These quantities are evaluated as a function of the mesh size N, on a Chebyshev collocation grid. Computation of the eigenvalues has been performed with EISPACK (see [22]). The upper part of the Table 1 corresponds to linear Lagrange elements with *exact* quadrature (i.e., FE preconditioning), whereas the results in the lower half were obtained with linear Lagrange elements and the trapezoidal quadrature rule for evaluation of the stiffness and mass matrices. This

TABLE 1

Finite-element preconditioning of various one-dimensional Dirichlet boundary value problems. $|\lambda_m|$ *and* $|\lambda_M|$ *are the lower and upper bounds of the eigenvalue spectrum of* $\hat{L}_{fe}^{-1} \cdot L_{ps}$; $\kappa = |\lambda_M|/|\lambda_m|$ *is the condition number.*

	$Lu = -u_{xx}$			$Lu = -\nabla((1+x^2)\nabla u)$			$Lu = -\nabla((1+10x^2)\nabla u)$		
N	$\lvert\lambda_m\rvert$	$\lvert\lambda_M\rvert$	κ	$\lvert\lambda_m\rvert$	$\lvert\lambda_M\rvert$	κ	$\lvert\lambda_m\rvert$	$\lvert\lambda_M\rvert$	κ
	Linear Lagrange elements with exact integration (FE preconditioning)								
4	7.146 (−1)	8.998 (−1)	1.26	5.988 (−1)	8.999 (−1)	1.50	2.241 (−1)	9.027 (−1)	4.03
8	7.776 (−1)	9.748 (−1)	1.24	7.470 (−1)	9.716 (−1)	1.30	5.639 (−1)	9.808 (−1)	1.74
16	7.446 (−1)	9.936 (−1)	1.33	7.658 (−1)	9.937 (−1)	1.30	6.547 (−1)	9.870 (−1)	1.51
32	7.233 (−1)	9.984 (−1)	1.38	7.209 (−1)	9.984 (−1)	1.38	6.985 (−1)	9.982 (−1)	1.43
64	7.085 (−1)	9.996 (−1)	1.41	7.060 (−1)	9.996 (−1)	1.42	6.950 (−1)	9.996 (−1)	1.44
128	7.009 (−1)	9.999 (−1)	1.43	6.988 (−1)	9.997 (−1)	1.43	6.936 (−1)	9.999 (−1)	1.44
	Linear Lagrange elements with trapezoidal quadrature (FD preconditioning)								
4	1.000 (0)	1.757 (0)	1.76	9.439 (−1)	1.560 (0)	1.65	3.983 (−1)	1.384 (0)	3.47
8	1.000 (0)	2.131 (0)	2.13	9.961 (−1)	2.040 (0)	2.05	8.583 (−1)	1.877 (0)	2.19
16	1.000 (0)	2.306 (0)	2.31	9.990 (−1)	2.268 (0)	2.27	9.851 (−1)	2.186 (0)	2.22
32	1.000 (0)	2.388 (0)	2.39	9.997 (−1)	2.372 (0)	2.37	9.962 (−1)	2.336 (0)	2.34
64	1.000 (0)	2.428 (0)	2.43	9.999 (−1)	2.421 (0)	2.42	9.990 (−1)	2.405 (0)	2.41
128	1.000 (0)	2.448 (0)	2.45	1.000 (0)	2.445 (0)	2.45	9.998 (−1)	2.437 (0)	2.44

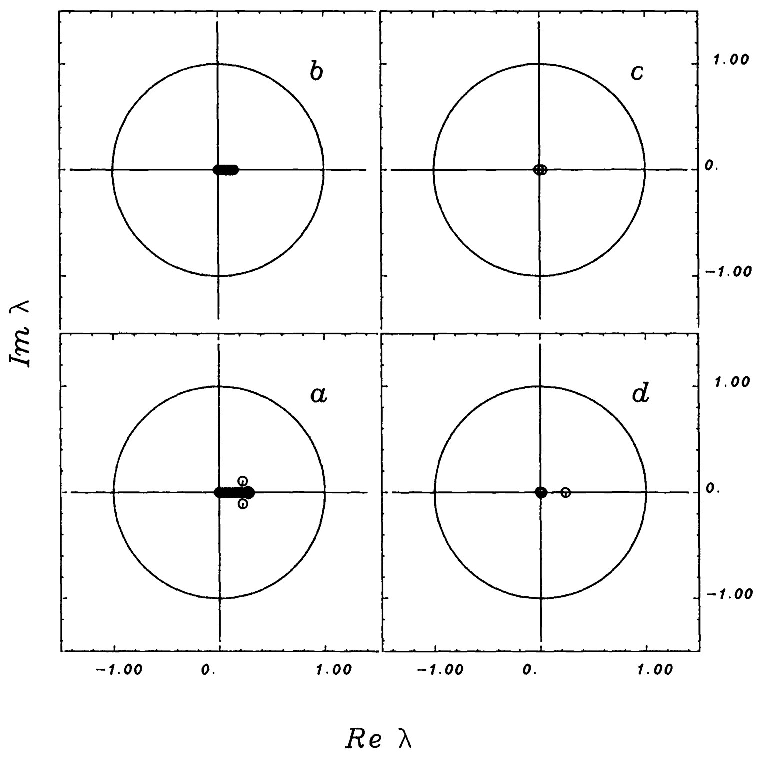


FIG. 4. *Spectrum of the iteration matrix for the problem* $-d^2u/dx^2+u$, *with Dirichlet conditions. Cases a, b, c, d correspond to linear* (L1), *quadratic* (L2), *cubic Lagrange* (L3), *and cubic Hermite* (H3) *finite elements, respectively.*

yields the classical three-point finite-difference stencil (five-point stencil in two-dimensional space) and will be referred to in the sequel as FD preconditioning.

Both preconditionings lead to a condition number independent of the grid size even for cases where $p(\bar{r})$ has important variations. The FD condition number tends toward $\pi^2/4$ ($\cong 2.47$), its asymptotic value, as shown in [17]. The FE value is even lower ($\cong 1.45$), a strong indication of the "closeness" of L_{fe} and L_{ps}.

Figures 4 and 5 show the eigenvalue spectrum of the iteration matrix (3.17) for the one-dimensional operators:

$$L \triangleq -\frac{d}{dx}\left((1+\mu x^2)\frac{d}{dx}\right)+I, \qquad x \in (-1,+1), \tag{4.1}$$

with Dirichlet boundary conditions and $\mu = 0$ (see Fig. 4) or $\mu = 100$ (see Fig. 5). Finite-element preconditioning is performed with linear (L1), quadratic (L2), and cubic (L3), Lagrange elements and with cubic Hermite (H3) elements. The results are displayed clockwise, from (a)-(d). Figures 4 and 5 were obtained with Chebyshev collocation and $N = 64$. When $\mu = 0$, the eigenvalues of the iteration matrix are close to zero, yielding fast convergence of the iterations. This is especially true for the L2, L3, and H3 elements. Increasing the operator complexity (i.e., $\mu = 100$) does not affect the L1 and H3 spectral properties. However, for the L2 and L3 preconditionings, we observe a progressive "blowup" of the spectrum in the complex plane, eventually leading the iterative scheme to diverge. This property, together with considerations on computational costs, concur to focus the study on L1 elements.

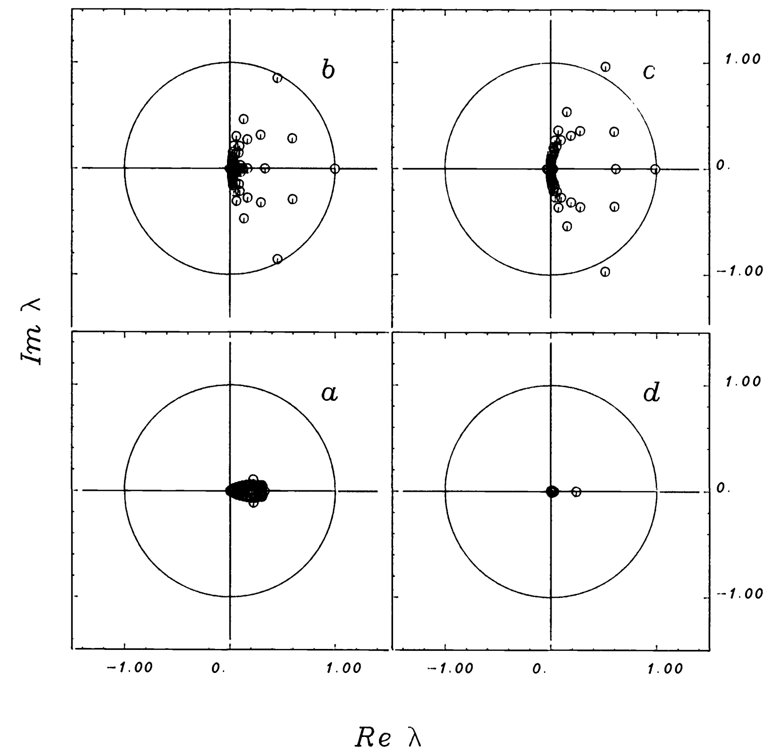


FIG. 5. *Same as Fig. 4 except for problem* $-d/dx((1+100x^2)\,du/dx)+u(x)$.

Let us indeed make a brief preliminary incursion into the discussion of convergence results. Table 2 displays error norms and (relative) CPU times for two Poisson equations in two space dimensions with Dirichlet boundary conditions on the unit square $(0,1)\otimes(0,1)$. The solution of the first problem (upper half) is $u(x,y)=\sin(\pi/2)x\cos 2\pi y$ and for the second problem (lower half), we have $u(x,y)=\sin 4\pi x\cdot\sin 4\pi y$. The CPU times are given in units of the L1 (16×16) initial guess (u^0) computation time. As seen in Table 2 (upper part), the L1 (16×16) converged result is only 0.25 more costly than the initial H3 (16×16) calculation and twice as cheap as its converged value. More striking perhaps are the results in the lower half of Table 2. The cutoff value of the Chebyshev representation of $\sin 4\pi x\cdot\sin 4\pi y$ with $N=16$ is too low to allow machine accuracy to be reached (we should therefore go to $N=32$). But the converged value with the L1(16×16) iterative scheme is still 20 times more accurate (in L^∞ error norm) than the L2(32×32) initial guess and costs only one fifth of its CPU time. This is due to the computation costs of static condensation or, more generally, to the presence of internal degrees of freedom. We shall come back to these arguments in the next paragraph.

It seems clear that, although the number of preconditioning iterations entailed by the L1 scheme might be higher than in the H3 scheme, the low computational cost per iteration of the L1 scheme makes it preferable.

Further evidence of the robustness of this scheme is given in Fig. 6 that displays the eigenvalues of (3.17) for the operator (4.1) with $\mu=100$ and Neumann boundary conditions. Here, collocation is performed on a Gauss-Lobatto-Legendre grid with $N=32$. Collocation on a Gauss-Lobatto-Chebyshev mesh would produce a similar spectrum. This operator is nearly singular because of the high value of $p(\bar r)$. As a consequence, a few eigenvalues are close to 1. This does not prevent, however, the main part of the spectrum from being inside the unit circle, not far from zero. Later on we shall verify on a particular problem that, even in this difficult case, the L1 scheme converges towards machine accuracy.

Figures 7-16 are all related to the L1 preconditioning (both FE and FD) of two-dimensional partial differential operators:

$$L \triangleq -\bar\nabla((1+\mu x^2y^2)\bar\nabla)+I, \qquad x,y\in(-1,1)\otimes(-1,1), \tag{4.2}$$

on Chebyshev grids with $N_x=N_y=16$ and homogeneous Dirichlet (or Neumann)

TABLE 2

Comparison of different types of finite-element preconditioning with respect to accuracy and computing times for two Dirichlet boundary value problems. The CPU *times are normalized with respect to the L1 initial guess for the coarser grid.*

	$\|u-u^0\|_2$	$\|u-u^{\text{conv}}\|_2$	$\|u-u^0\|_\infty$	$\|u-u^{\text{conv}}\|_\infty$	CPU t u^0	CPU t u^{conv}	No iterations
			Problem $u(x,y)=\sin(\pi/2)x\cdot\cos 2\pi y$				
L1 (16×16)	2.27 (−3)	1.65 (−13)	8.21 (−3)	6.57 (−13)	1	22.8	10
L2 (16×16)	3.46 (−6)	2.54 (−14)	1.33 (−5)	7.76 (−14)	5.5	31.0	5
H3 (16×16)	1.13 (−5)	8.07 (−14)	2.93 (−5)	2.35 (−13)	16.8	36.3	5
			Problem $u(x,y)=\sin 4\pi x\cdot\sin 4\pi y$				
L1 (16×16)	3.07 (−2)	6.37 (−7)	9.88 (−2)	3.86 (−6)	1	13.5	6
L2 (16×16)	4.40 (−4)	8.01 (−8)	1.71 (−3)	2.20 (−7)	5.5	18.8	3
L1 (32×32)	8.29 (−3)	9.09 (−15)	2.82 (−2)	2.91 (−14)	9.5	138.5	11
L2 (32×32)	2.30 (−5)	9.08 (−15)	8.25 (−5)	2.91 (−14)	65.3	384.5	5

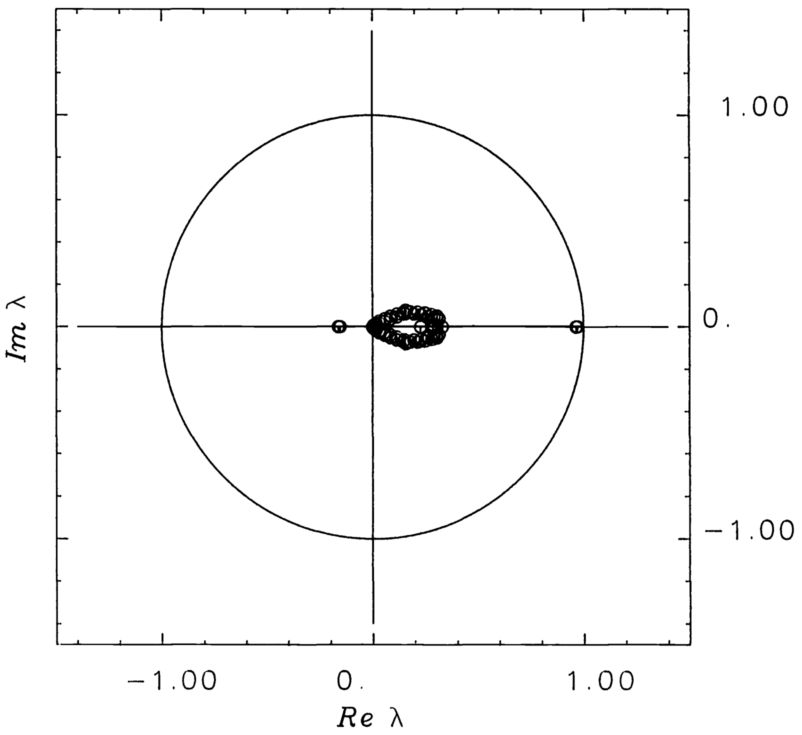


FIG. 6. *Spectrum of iteration matrix for problem* $-d/dx((1+100x^2)\,du/dx)+u(x)$ *with Neumann conditions. The preconditioner uses linear elements.*

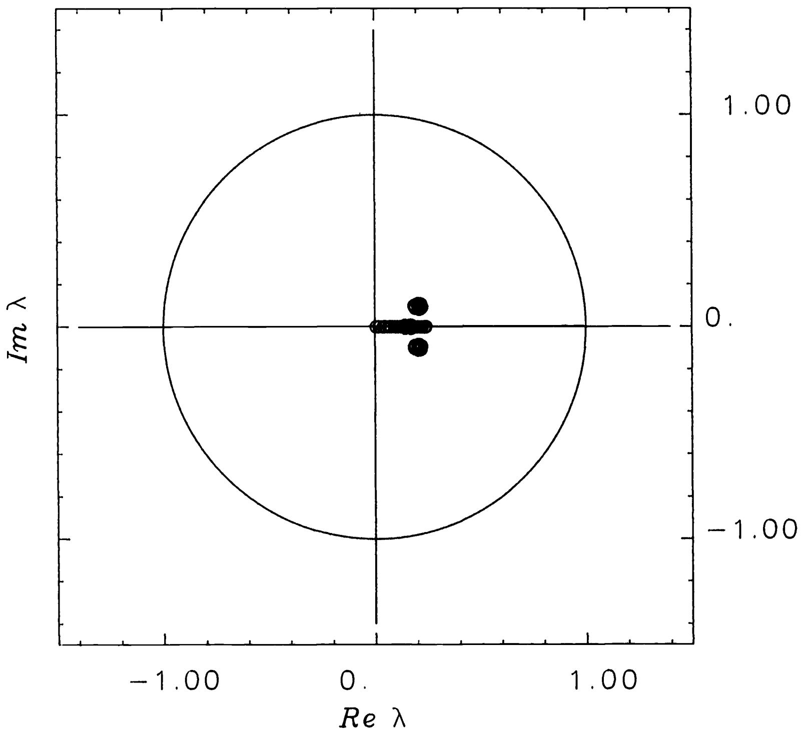


FIG. 7. *Spectrum of iteration matrix for problem* $-\Delta u+u$ *with Dirichlet conditions. Bilinear finite elements are used in the preconditioning.* $N_x = N_y = 16$.

boundary conditions. Figures 7, 8, 10, and 11 correspond to FE preconditioning of problems with essential boundary conditions and $\mu = 0, 1, 10$, and 100, respectively.

Once again in *all cases*, the spectrum of the iteration matrix (3.17) lies inside the unit circle, ensuring convergence of (3.16) with a relaxation factor $\alpha_k = 1$ $(k = 1, \cdots)$. For $\mu = 1$ the spectral radius of A is 0.32. Inserting this value into (3.18) gives the number of iterations required for an error reduction of, say, $\zeta = 10^{-10}$: $n \cong 18$. In most cases we should expect less than 30 iterations to reach machine accuracy.

Figure 9 displays the spectrum of the iteration matrix in a case where the preconditioning operator has been chosen as the FE approximation of an operator $\mathscr{L} \neq L$. Basically, the differential problem (4.2) with $\mu = 1$ and Dirichlet conditions is preconditioned with the FE discretization of (4.2) with $\mu = 0$. The stability of the eigenvalues inside the unit circle is quite remarkable.

Spectral properties of FE preconditioning of Neumann problems do not differ essentially from those of Dirichlet problems. Figures 12 and 13 are the exact counterpart of Figs. 8 and 9, except for the boundary conditions. The conclusions that can be drawn are the same.

We shall now pay some attention to L1 FD preconditioning. Figures 14–16 display the eigenvalues of the iteration matrix (3.17) for two differential problems (4.2) with Dirichlet boundary conditions and $\mu = 0$ (Fig. 14) or $\mu = 1$ (Figs. 15 and 16).

The differences with FE preconditioning are striking: in this case, the spectrum straddles the unit circle and underrelaxation is compulsory to ensure convergence of the iterations. Figures 15 and 16 show the effects of changing the reference operator for the preconditioner. If, again, the preconditioner is the FD discretization of (4.2)

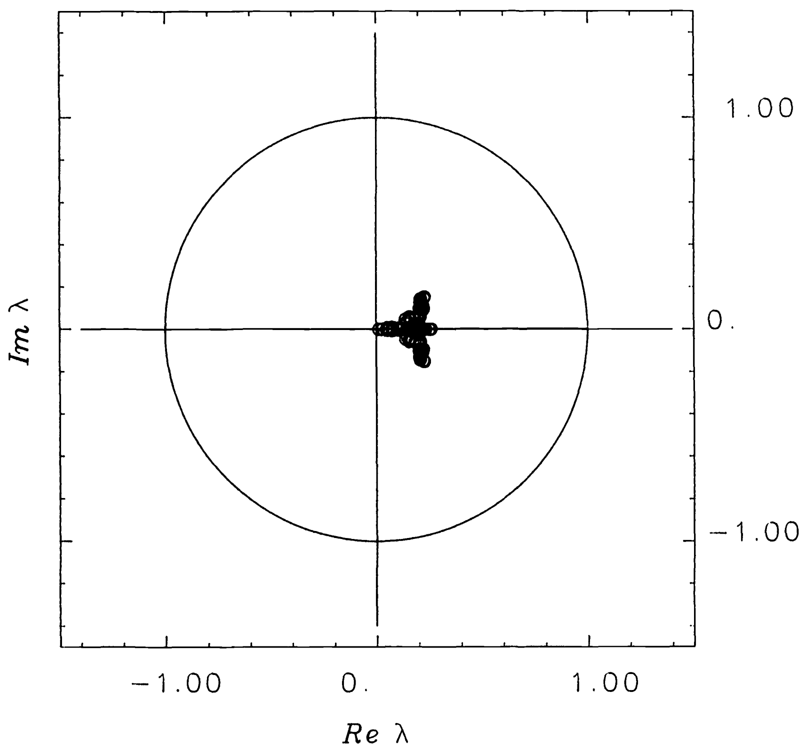


FIG. 8. *Same legend as in Fig. 7 except the problem is* $-\bar{\nabla}((1+x^2y^2)\bar{\nabla}u)+u$.

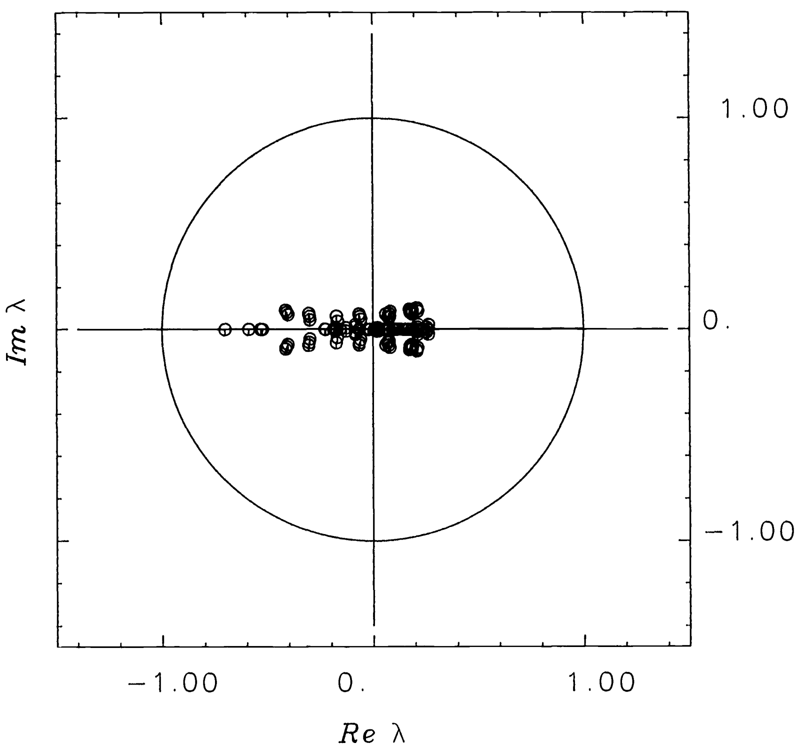


FIG. 9. *Same legend as in Fig. 8. The preconditioning operator is* $-\Delta u + u$.

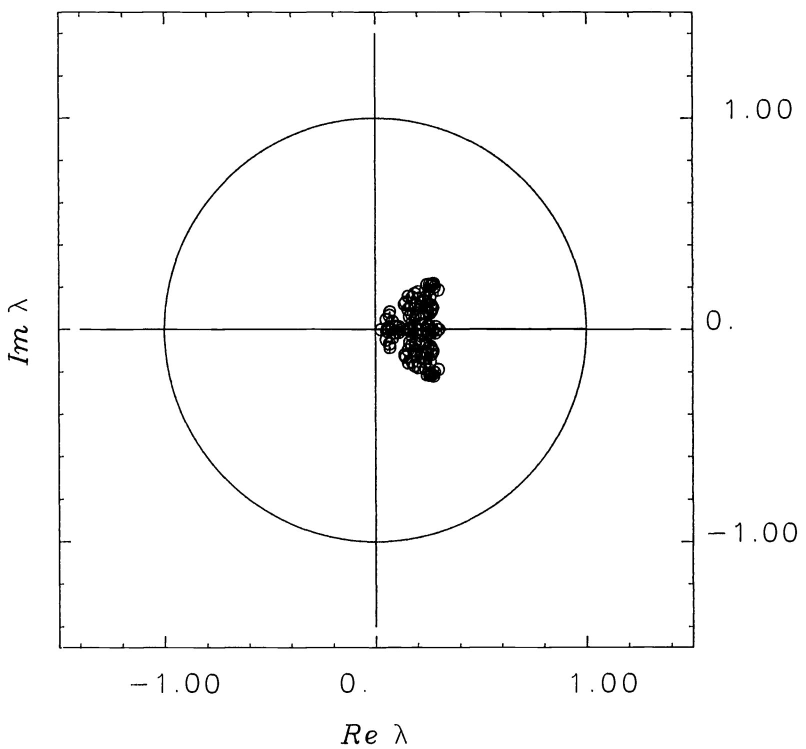


FIG. 10. *Same legend as in Fig. 7 except the problem is* $-\bar{\nabla}((1+10x^2y^2)\bar{\nabla}u)+u$.

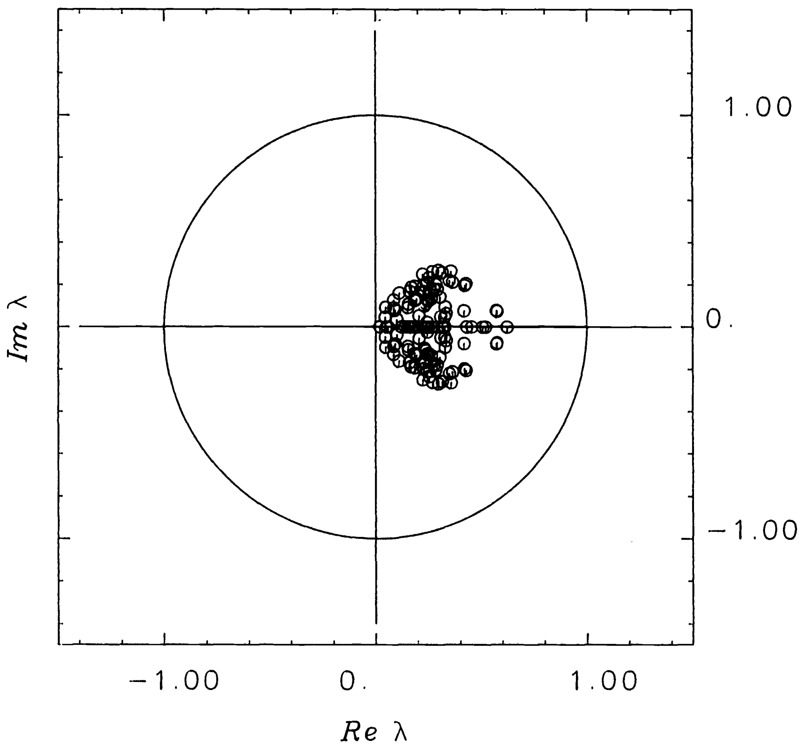


FIG. 11. *See Fig. 7 except the problem is* $-\bar{\nabla}((1+100x^2y^2)\bar{\nabla}u)+u$.

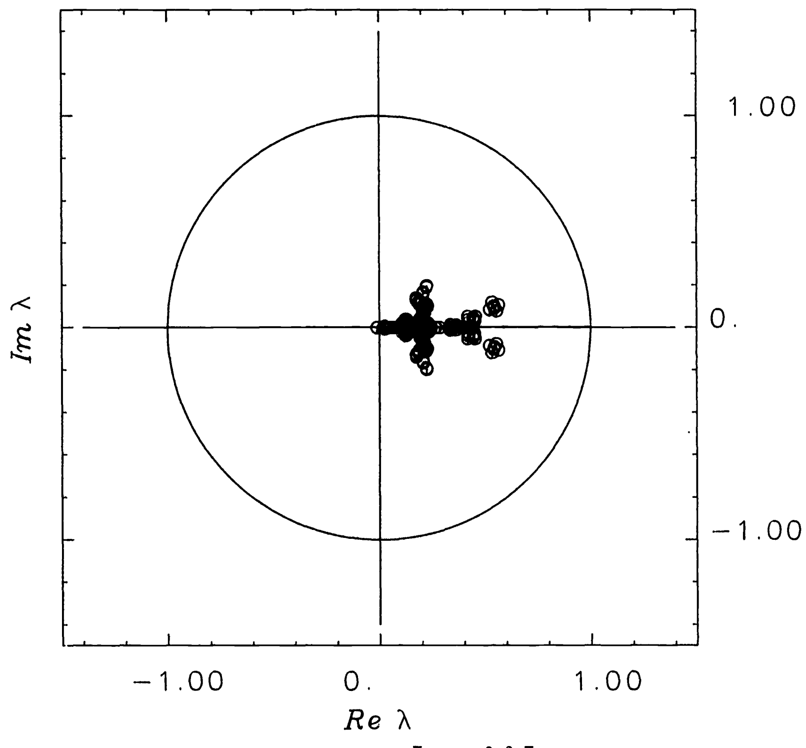


FIG. 12. *Spectrum of iteration matrix for problem* $-\bar{\nabla}((1+x^2y^2)\bar{\nabla}u)+u$ *with Neumann conditions. Bilinear finite elements are used in the preconditioner.* $N_x = N_y = 16$.

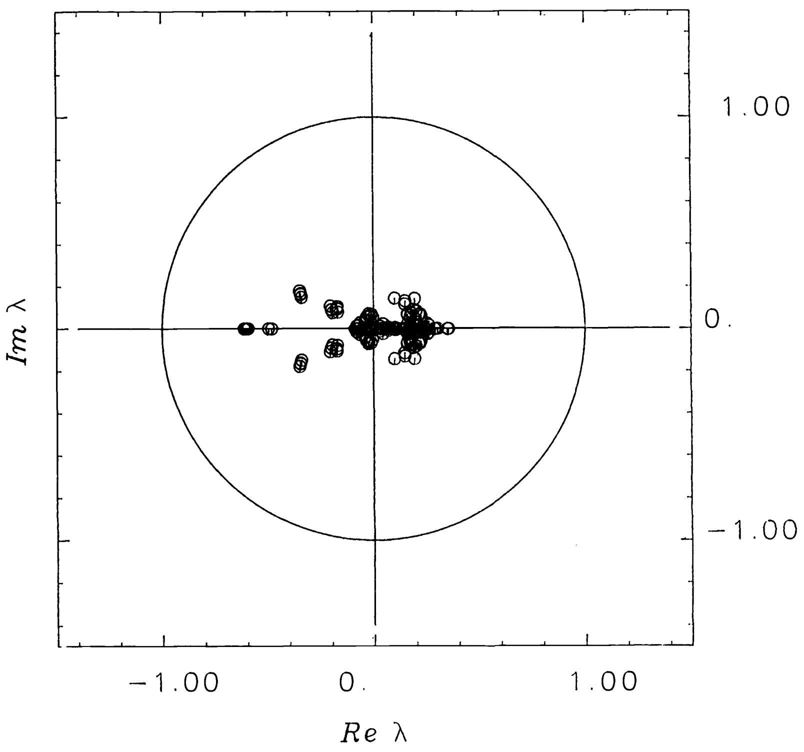


FIG. 13. *Same caption as in Fig. 12. The preconditioning operator is* $-\Delta u + u$.

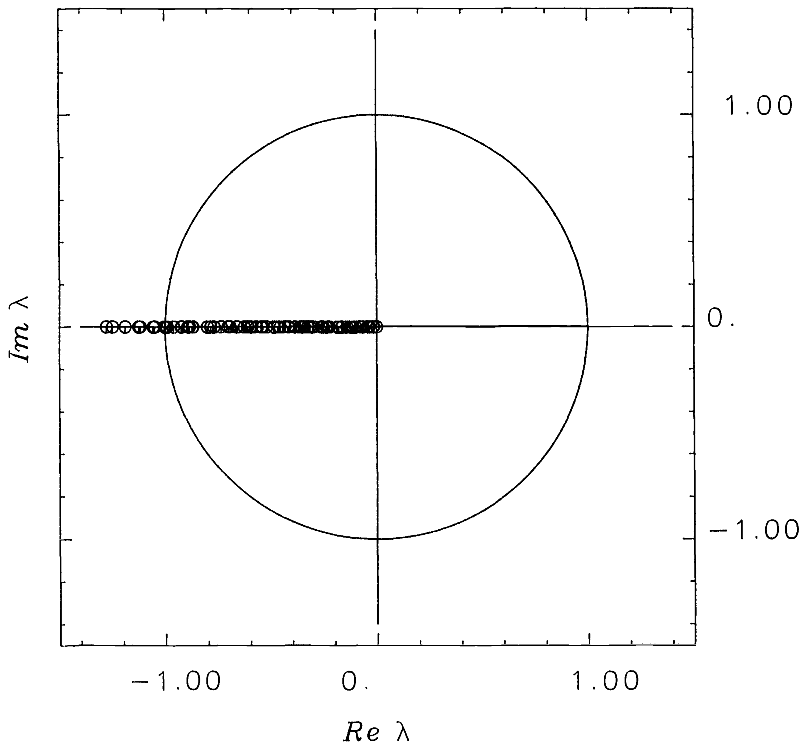


FIG. 14. *Spectrum of the iteration matrix for problem* $-\Delta u + u$ *with Dirichlet conditions. Finite differences constitute the preconditioner.* $N_x = N_y = 16$.

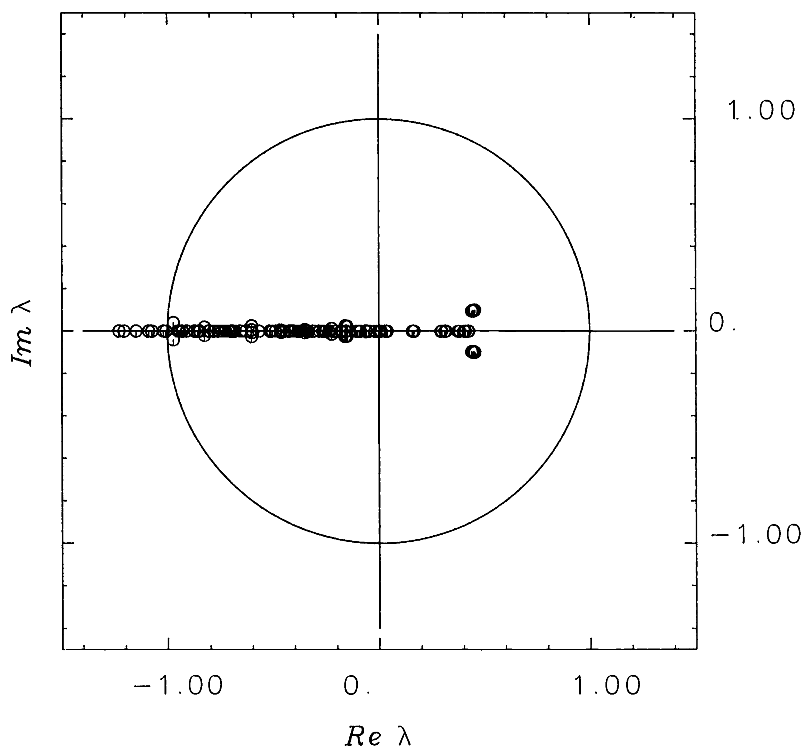


FIG. 15. *Same legend as in Fig.* 14 *except the problem is* $-\bar{\nabla}((1+x^2y^2)\bar{\nabla}u)+u$.

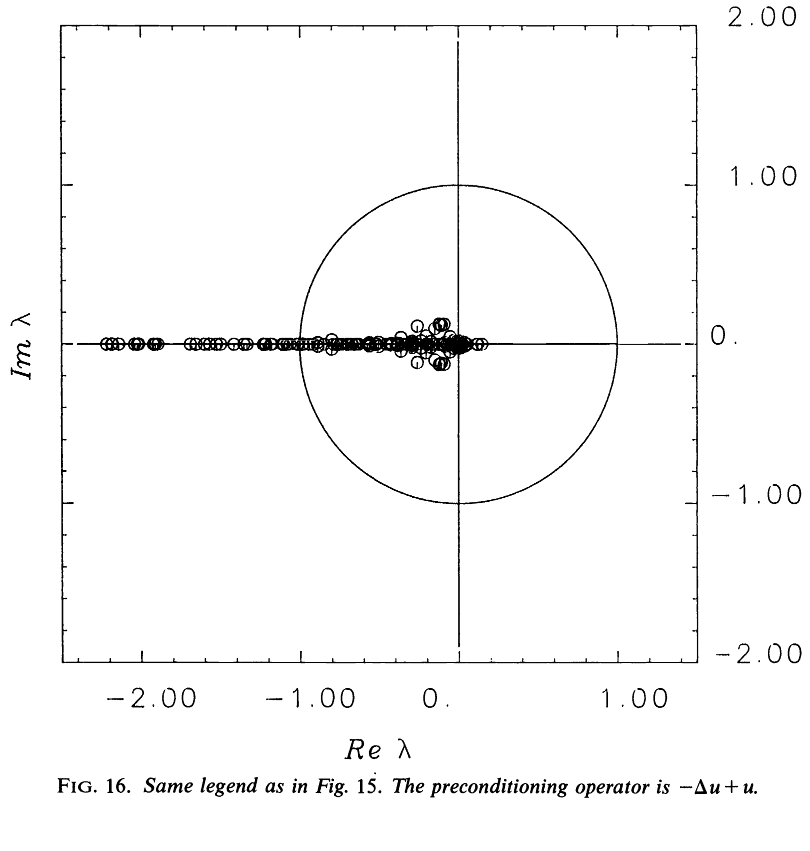


FIG. 16. *Same legend as in Fig.* 15. *The preconditioning operator is* $-\Delta u+u$.

with $\mu = 0$ (see Fig. 16), we observe an increase in the dispersion of the eigenvalues that will only alter the rate of convergence. Changing the nature of the collocation grid does not modify these results significantly.

Let us now examine the preconditioning iterations for the pseudospectral approximation of

$$Lu(x, y) = -\bar{\nabla}((1+x^2y^2)\bar{\nabla}u(x, y)) + u(x, y) = f(x, y), \qquad x, y \in (-1, 1)\otimes(-1, 1) \tag{4.3}$$

where the right-hand side $f(x, y)$ is such that

$$u(x, y) = \sin^2 \pi x \sin^2 \pi y \exp(x+y). \tag{4.4}$$

Both homogeneous Dirichlet and Neumann boundary conditions have been treated according to the schemes (3.14)-(3.16) and (3.21)-(3.24), respectively, with L1 elements on a Chebyshev grid ($N_x = N_y = 32$).

Figures 17 and 18 show the evolution of discrete L^∞ error norm $|||\varepsilon|||_\infty$ as a function of the iteration index. The circles correspond to FE preconditioning and the squares to FD preconditioning. While in Fig. 17 the preconditioned and preconditioner are the same operators, Fig. 18 exhibits the results when the preconditioner is obtained from (4.2) with $\mu = 0$. In all cases the optimum relaxation factors have been introduced from careful study of the spectrum.

For FE preconditioning, the introduction of the optimum relaxation factor does not change the convergence speed very much; a few iterations at most. For FD preconditioning, it is essential as already mentioned. When both operators are equal (preconditioned and preconditioner), FE preconditioning requires less than half the

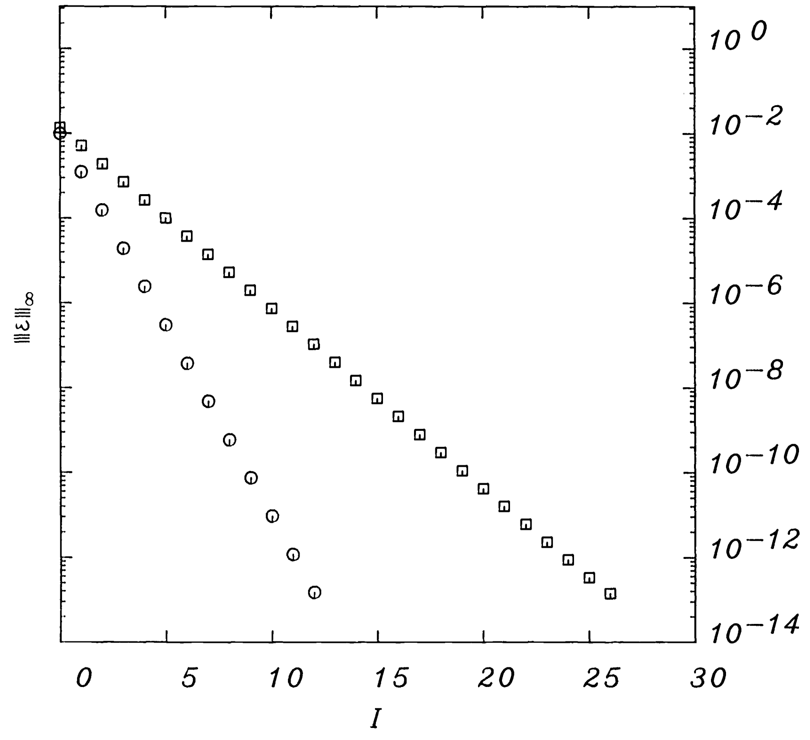


FIG. 17. *Plot of the error in maximum norm with respect to the number of iterations for the problem* $-\bar{\nabla}((1+x^2y^2)\bar{\nabla}u)+u=f$, *subject to homogeneous Dirichlet conditions. The exact solution is* $u(x, y) = \sin^2\pi x \cdot \sin^2\pi y \cdot \exp(x+y)$. *Here,* $N_x = N_y = 32$. *The dots correspond to finite-element preconditioning with* $\alpha_k = 1.13$, *while squares deal with finite difference preconditioning and* $\alpha_k = 0.62$, $(k = 1, \cdots)$.

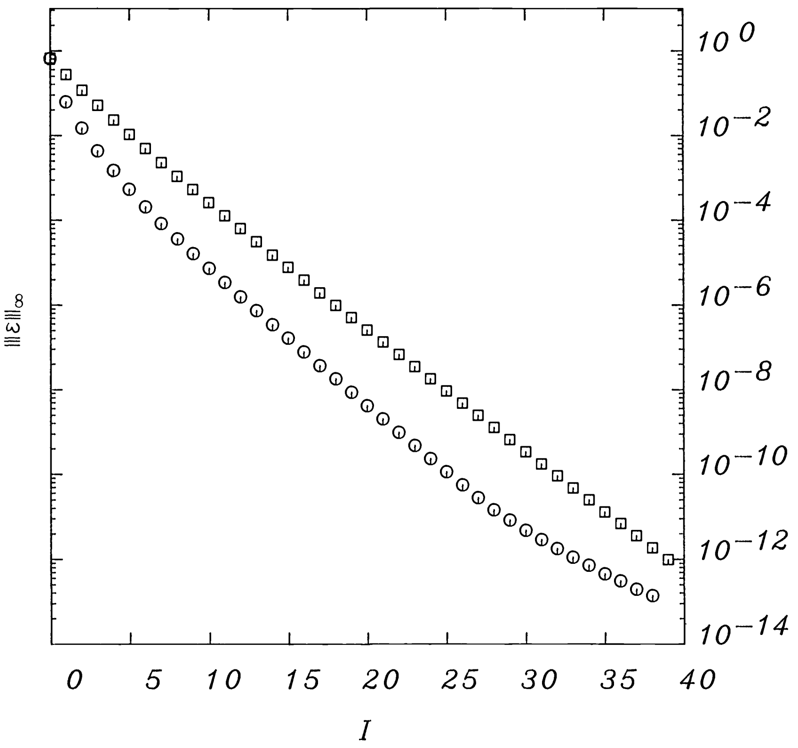


FIG. 18. *Same legend as Fig. 17. The preconditioning operator is $-\Delta u + u$. The dots of the* FE *preconditioning are obtained with $\alpha_k = 0.75$ and the squares of the* FD *preconditioning with $\alpha_k = 0.45$, $(k = 1, \cdots)$.*

number of FD iterations to reach roundoff error. We also remark that the estimation of 18 iterations made earlier is well verified. By changing the preconditioner, we only multiply this result by a factor two. Figures 19 and 20 present the same results as Figs. 17 and 18, but for the Neumann problem; the conclusions are the same.

As a last example of the effectiveness of the L1 FE preconditioning, we turn back to a one-dimensional Neumann problem:

$$Lu(x,y) \triangleq -\frac{d}{dx}\left((1+100x^2)\frac{du}{dx}\right) + u(x) = f(x), \qquad x \in (-1,+1),$$
$$u'(-1) = u'(1) = 0, \tag{4.5}$$

whose iteration matrix has the spectrum displayed in Fig. 6. The right-hand side (4.5) is such that $u(x) = (x-1)^2(x+1)^2 \exp(x)$. Convergence must be slow, as already stated (see Figs. 21 and 22).

The preconditioning of Neumann problems involves the projection operator $I_{n,h}^{L,0}$ for the residuals of the equation. Use of this operator gives a strong argument for the convergence of the iterative scheme (see (3.26)-(3.28) and the related discussion). However, the numerical calculations indicate that the inclusion of these residuals at Neumann boundary nodes might enhance the convergence. This is shown in Fig. 21 where problem (4.5) is solved on a Legendre grid with $N = 32$ and the relaxation factor $\alpha_k = 1$, $(k = 1, \cdots)$. If the residuals $(L_{ps}u^k - f)$ are projected onto the interior domain, there is a sudden drop in the error and afterwards the convergence is extremely slow (see squares). If, however, we use these residuals both in the domain and at the Neumann boundary nodes, convergence is increased and the 10^{-9} error level reached in 180 iterations.

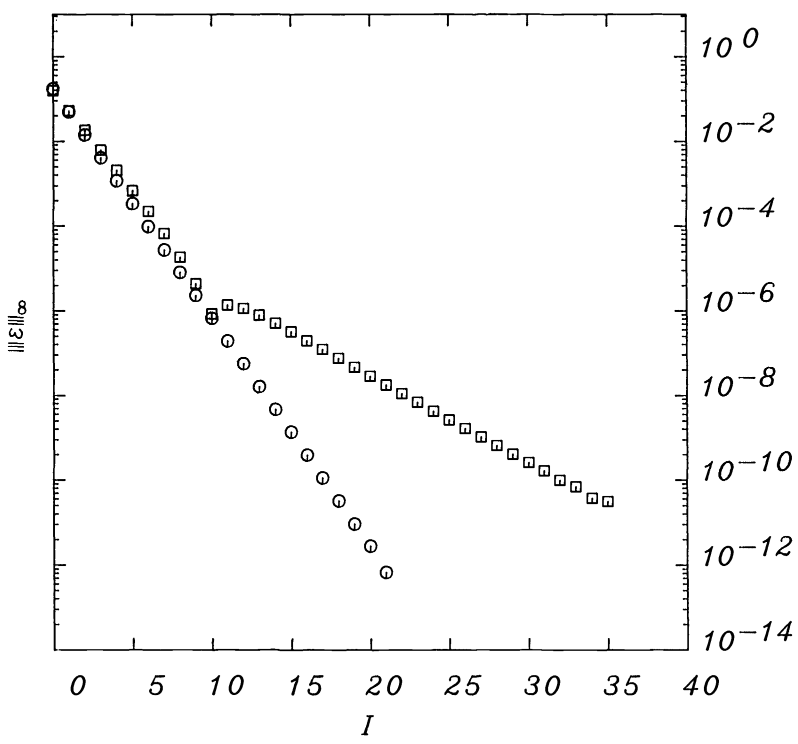


FIG. 19. *Same legend as in Fig. 17. Here, Neumann homogeneous boundary conditions are applied. The* α_k *values are* 1.30 *and* 0.65 *for* FE *and* FD *preconditioning, respectively.*

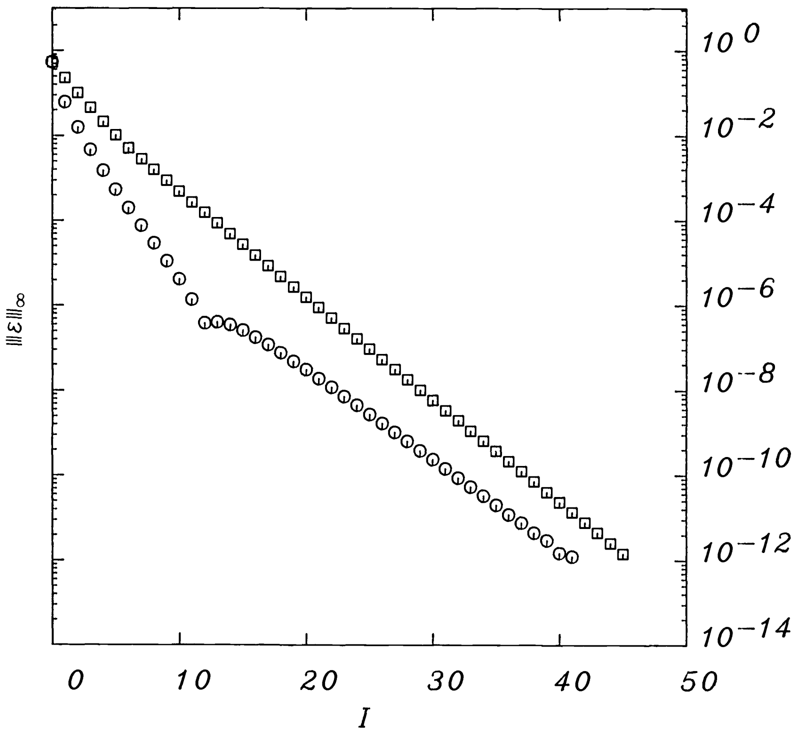


FIG. 20. *See caption of Fig. 19. The preconditioning operator is* $-\Delta u+u$. *The* α_k *values are* 0.80 *and* 0.40 *for* FE *and* FD *preconditioning, respectively.*

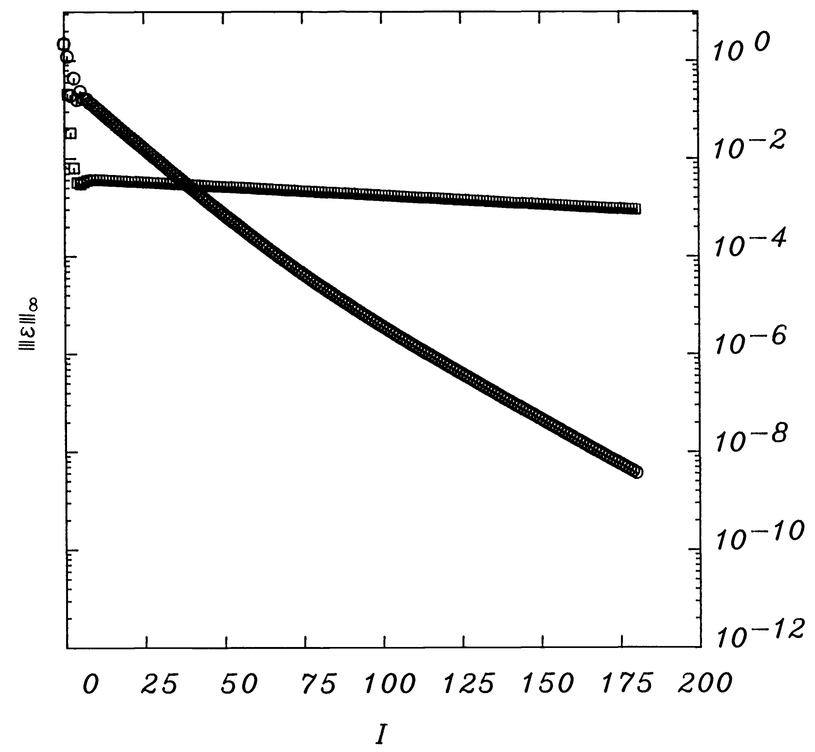


FIG. 21. *Error in maximum norm versus the number of iterations. The problem* $-d/dx((1+100x^2)\,du/dx)+u=f$ *with Neumann homogeneous boundary conditions is solved on* $(-1,1)$. *The exact solution is* $u(x)=(x-1)^2(x+1)^2\exp(x)$. $N_x=N_y=32$. *Linear finite-element preconditioning of a Legendre pseudospectral approximation is used with* $\alpha_k=1$. *The convergence curve with dots occurs when at the boundaries, the residuals to the* pde *and the boundary conditions are incorporated. Squares correspond to the case where at the boundaries the residual to the conditions are the only ones taken into account.*

Figure 22 displays the error norms with the relaxation factors $\alpha_k=1$ (circles) and $\alpha_k=1.307$ (squares) which is the optimum value.

As a final comment, let us examine again the question of operation counts. We denote by X the number of preconditioning iterations equivalent to a direct solution of the pseudospectral linear system (2.16). Given the fact that the *actual* number of iterations is approximately 20, let R be the computation time reduction factor. Using the notation defined previously, we have:

$$X(N)=\frac{(w_{ps}(N)-w_{fe}(N))+(r_{ps}(N)-r_{fe}(N))}{r_{fe}(N)+2N^2\log_2 N}\cong O(N^3), \tag{4.6}$$

$$R(N)=\frac{w_{ps}(N)+r_{ps}(N)}{w_{fe}(N)+r_{fe}(N)+20(r_{fe}(N)+2N^2\log 2N)}\cong O(N^2) \tag{4.7}$$

where N is the cutoff of the Chebyshev expansion in one dimension. Figure 23 represents the dependence of these quantities with respect to N. Of course, the direct solution of the pseudospectral system is out of question because of both huge operational count and bad numerical conditioning. It is, however, interesting to realize that the computation reduction factor might go as high as 100 for $N\cong 30$.

5. Spectral IDeC for the finite-element solution of singular problems. The preconditioned pseudospectral method may be regarded from an other point of view, i.e., as an application of the iterative Defect Correction (IDeC) principle proposed by Auzinger

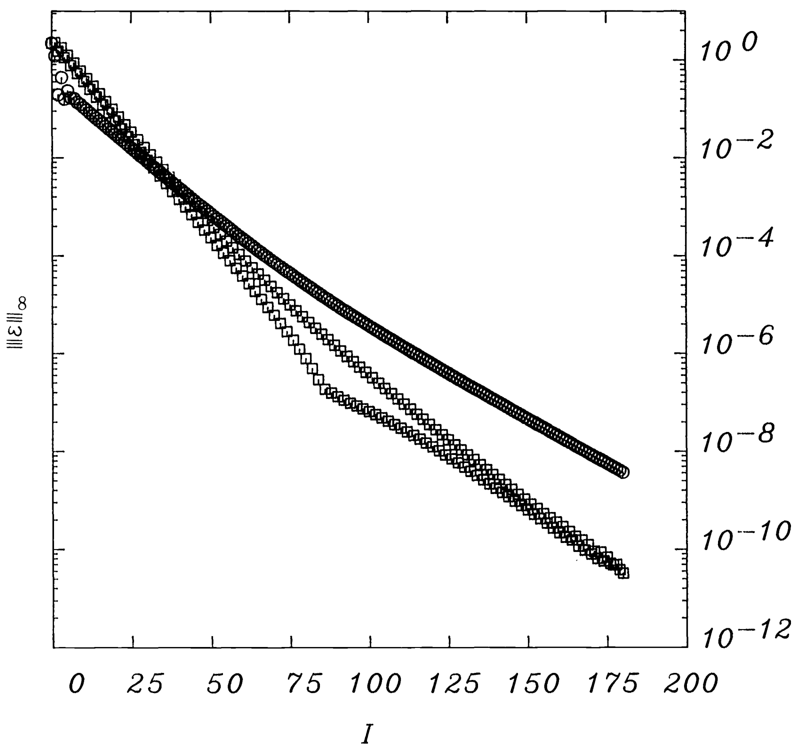


FIG. 22. *See caption of Fig.* 21. *Both the residuals to the* pde *and the boundary conditions are considered at the boundaries. The curve with dots is obtained with* $\alpha_k = 1.0$, *while the curve with squares is gotten for* $\alpha_k = 1.307$, $(k = 1, \cdots)$.

and Stetter [1], [23]. The defect correction works as follows. Suppose we want to solve the problem

$$L_{ps}\bar{u} = \bar{f}. \tag{5.1}$$

However, we possess a very efficient procedure for the related problem

$$\hat{L}\bar{v} = \bar{f}. \tag{5.2}$$

Therefore, the following iterative procedure yields the solution

$$\bar{u}^{k+1} = \bar{u}^k - \hat{L}^{-1}(L_{ps}\bar{u}^k - \bar{f}). \tag{5.3}$$

In this last relation, we recognize (2.27). It is well known that the convergence of this process depends on the contractivity of the operator:

$$I - \hat{L}^{-1}L_{ps}. \tag{5.4}$$

In the previous section, we analyzed the properties of (5.4) with $\hat{L}$ being a finite-element approximation of the original problem with various interpolation techniques. As a consequence, we may consider that spectral accuracy may be achieved from standard finite-element codes with a little extra effort of programming. This requires only the evaluation of the pseudospectral residual.

A malicious objection that tempers the enthused spectral numericist comes from the presence of singularities that dramatically alter the rate of converge of the numerical

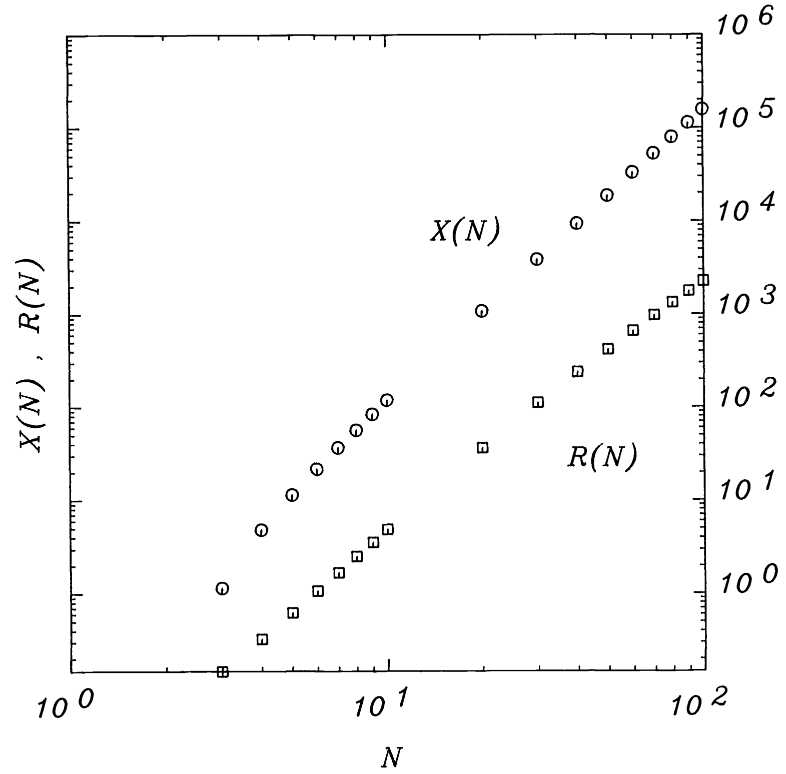


FIG. 23. *With respect to the number of degrees of freedom N, X(N) yields the number of preconditioning iterations equivalent to the direct solution of the pseudospectral linear system. R(N) is the speedup factor achieved by the actual computations where convergence is attained in about* 20 *iterations.*

method. In order to face the argument, we will revisit the following problem (see [16], [18]):

$$-\Delta u = 1, \qquad \Omega = (-1, 1) \otimes (-1, 1) \tag{5.5}$$

with homogeneous Dirichlet conditions on $\partial\Omega$. The geometric singularities arise near the four corners, where the problem (5.5) is solved while the boundary conditions lead to a vanishing Δu. The local behavior of the solution is

$$u = O(r^2|\ln r|) \quad \text{as } r \to 0, \tag{5.6}$$

where r is the distance from a corner. The exact solution is a double cosine series given by (see [16])

$$u(x, y) = \frac{64}{\pi^2} \sum_{\substack{n=1 \\ n \text{ odd}}}^{\infty} \sum_{\substack{m=1 \\ m \text{ odd}}}^{\infty} (-1)^{1/2(n+m)+1} \left(\cos \frac{n\pi x}{2} \cos \frac{n\pi y}{2} \Big/ nm(n^2 + m^2) \right)$$

The numerical evaluation of this expression was performed by a singular finite-element technique due to Descloux and Tolley [8]. From the analysis carried out by Canuto and Quarteroni [4], we know that the spectral solution of a Dirichlet boundary value problem converges as $O(N^{1-s})$ in Sobolev norm $\|\cdot\|_1$, where s is the highest order of derivative still belonging to a square integrable weighted space and N, the inverse of the one-dimensional mesh. In our case, $s=2$. The results of the above example (see Table 3) are presented in discrete L^2 and L^∞ norms defined by the relations

$$|||\varepsilon|||_2 \triangleq \frac{1}{N^2}\left[\sum_{i=1}^{N^2} (\bar{u}_i - u_i)^2\right]^{1/2}, \qquad |||\varepsilon|||_\infty \triangleq \max_i |\bar{u}_i - u_i|,$$

with $\bar{u}_i$ the computed values and u_i the exact solution evaluated on G_N.

TABLE 3
Discrete L^2 and L^∞ errors norms for the FE *guess u^0, the converged pseudospectral solution u^{conv} and the Tau solution (see [16]) of problem $-\Delta u = 1$.*

N	u^0		u^{conv}		Tau
	$\lvert\lvert\lvert\varepsilon\rvert\rvert\rvert_2$	$\lvert\lvert\lvert\varepsilon\rvert\rvert\rvert_\infty$	$\lvert\lvert\lvert\varepsilon\rvert\rvert\rvert_2$	$\lvert\lvert\lvert\varepsilon\rvert\rvert\rvert_\infty$	$\lvert\lvert\lvert\varepsilon\rvert\rvert\rvert_\infty$
8	2.82 (−4)	7.37 (−3)	6.91 (−7)	1.61 (−5)	-
16	3.94 (−5)	1.83 (−3)	9.65 (−9)	7.47 (−7)	3.52 (−5)
32	5.23 (−6)	4.56 (−4)	1.56 (−10)	5.52 (−8)	2.23 (−6)

From Table 3, we observe that the error $|||\varepsilon|||_\infty$ on u^0 converges as N^{-2} as it should (see [6]), whereas, the $|||\varepsilon|||_2$ error on u^{conv} converges algebraically as N^{-6} which is the same rate as the Tau method. Nonetheless, the $|||\varepsilon|||_\infty$ error produced by the pseudospectral calculation is consistently 40 times smaller than the $|||\varepsilon|||_\infty$ error given by the Tau method. This different behavior is attributed to the fact that the boundary conditions are exactly enforced in the pseudospectral approach, whereas interior collocation points deal with the residual to the pde. The Tau approach is more global and the boundary conditions are affected by the solution of the full system. Even if this is by no means spectral accuracy, the rate of convergence is still faster than the convergence shown by classical techniques for singular problems (see [18], [19]). The N^{-6} decay rate of the error exceeds by a few orders of magnitude the theoretical estimate, which in this case is too pessimistic.

We can estimate the power of the IDeC pseudospectral algorithm estimating the grid size necessary for a finite-element calculation to achieve the same level of accuracy as the collocation results. Instead of $N=8$, 16, 32, we get the values $\bar{N}=171$, 791, 3000, respectively. The computational work for the finite element with the $\bar{N}$ discretization requires

$$W_{\text{fe}}(\bar{N}) = w_{\text{fe}}(\bar{N}) + r_{\text{fe}}(\bar{N}) = \bar{N}^4 + 2\bar{N}^3 + O(N^2),$$

while the spectral IDeC algorithm performs

$$W_{\text{ps}}(N) = w_{\text{fe}}(N) + r_{\text{fe}}(N) + \bar{x}(2N^2 \log 2N + r_{\text{fe}}(N)) \tag{5.7}$$

operations. In the previous expression, $\bar{x}$ is the number of iterations required to obtain convergence. Typically, we observed that $\bar{x}=20$ for this example. Consequently, to achieve the best accuracy, the speedup factor of the pseudospectral scheme with respect to the finite-element computation is

$$\frac{W_{\text{fe}}(\bar{N})}{W_{\text{ps}}(N)}.$$

For $N=16$ and $\bar{N}=791$, the speedup factor is 1.35 10^6. Of course, no one would choose to solve the finite-element problem with the $\bar{N}$ discretization, but the performance gain provided by the collocation procedure is striking.

To conclude this section, we compare the computational work of the Tau method with respect to the preconditioned pseudospectral technique. If we use the Haidvogel and Zang algorithm [16] with a one-dimensional diagonalization and the solution of tridiagonal systems, we obtain the following operations count (see [17]):

$$W_{\mathrm{Tau}}(N) = O(N^3).$$

This count is in favor of the Tau method if it is compared to (5.7). But the Tau algorithm works essentially for equations with constant coefficients on very simple geometries. The extension to complicated geometries and the nonconstant coefficients case seems to be a formidable task.

6. Conclusions. In this paper, we derived the analytical expressions of the pseudospectral method based on general Jacobi polynomials. The collocation grid uses Gauss–Lobatto quadrature nodes. As the condition number of the pseudospectral matrix system for elliptic problems increases with the fourth power of the number of degrees of freedom, a preconditioning technique resorting to finite-element computation is analyzed. Among the various possible interpolants, the Lagrangian bilinear element shows extremely good properties from both the aspects of convergence and computational efficiency. The finite-element framework for the preconditioner provides the user with a powerful tool to treat either Dirichlet or Neumann boundary conditions. Even in the presence of geometric singularities, the preconditioned pseudospectral approach performs very well and far better than theoretical estimates predict.

Acknowledgments. We are indebted to C. Canuto and A. Quarteroni for several discussions that helped resolve open questions in the course of this research.

REFERENCES

[1] W. Auzinger and H. J. Stetter, *Defect Corrections and Multigrid Iterations*, Lecture Notes in Mathematics 960, W. Hackbusch and U. Trottenberg, eds., Springer-Verlag, Berlin, New York, 1982, pp. 327-351.

[2] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. Zang, *Spectral Methods in Fluid Dynamics*, Springer-Verlag, Berlin, New York, 1988.

[3] C. Canuto and P. Pietra, *Boundary and interface conditions within a finite element preconditioner for spectral methods*, Report 555, IAN-CNR, Pavia, 1987.

[4] C. Canuto and A. Quarteroni, *Variational methods in the theoretical analysis of spectral approximations*, 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, pp. 55-78.

[5] ———, *Preconditioner minimal residual methods for Chebyshev spectral calculations*, J. Comput. Phys., 60 (1985), pp. 315-337.

[6] P. G. Ciarlet, *The Finite Element Method for Elliptic Problems*, North-Holland, Amsterdam, 1979.

[7] P. J. Davis and P. Rabinowitz, *Methods of Numerical Integration*, Academic Press, New York, 1975.

[8] J. Descloux and M. D. Tolley, *Approximation of the Poisson problem and the eigenvalue problem for the Laplace operator by the method of the large singular finite element*, Res. Report 81-01, Seminär für Angewandte Mathematik, ETH Zentrum, Zürich, 1981.

[9] M. Deville and G. Labrosse, *An algorithm for the evaluation of multidimensional (direct and inverse) discrete Chebyshev transforms*, J. Comput. Appl. Math., 8 (1982), pp. 293-304.

[10] M. Deville and E. Mund, *Chebyshev pseudospectral solution of second-order elliptic equations with finite element preconditioning*, J. Comput. Phys., 60 (1985), pp. 517-533.

[11] ———, *On a Mixed One Step/Chebyshev Pseudospectral Technique for the Integration of Parabolic Problems Using Finite Element Preconditioning*, Lecture Notes in Mathematics, 1171, C. Brezinski et al., eds., Springer-Verlag, Berlin, New York, 1985, p. 399-407.

[12] G. H. Golub and C. F. Van Loan, *Matrix Computations*, North Oxford Academic, Oxford, 1983.

[13] D. Gottlieb and L. Lustman, *The spectrum of the Chebyshev collocation operator for the heat equation*, SIAM J. Numer. Anal., 20 (1983), pp. 909-921.

[14] D. GOTTLIEB, M. Y. HUSSAINI, AND S. A. ORSZAG, *Theory and Applications of Spectral Methods*, 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, pp. 1-54.

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

[16] D. B. HAIDVOGEL AND T. ZANG, *The accurate solution of Poisson's equation by expansion in Chebyshev polynomials*, J. Comput. Phys., 30 (1979), pp. 167-180.

[17] P. HALDENWANG, G. LABROSSE, S. ABBOUDI, AND M. DEVILLE, *Chebyshev* 3-D *spectral and* 2-D *pseudospectral solvers for the Helmholtz equation*, J. Comput. Phys., 55 (1984), pp. 115-128.

[18] J. P. HENNART AND E. H. MUND, *Singularities in the finite element approximation of two-dimensional diffusion problems*, Nuclear Sci. Engrg., 62 (1977), pp. 55-68.

[19] ———, *On the h- and p-version of the extrapolated Gordon's projector with applications to elliptic equations*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 773-791.

[20] Y. MORCHOISNE, *Résolution des équations de Navier-Stokes par une méthode pseudospectrale en espace-temps*, La Recheche Aérospatiale, 1979-5 (1979), pp. 293-305.

[21] S. A. ORSZAG, *Spectral methods for problems in complex geometries*, J. Comput. Phys., 37 (1980), pp. 70-92.

[22] B. T. SMITH et al., *Matrix Eigensystem Routines*—EISPACK *Guide*, Lecture Notes in Computer Sciences, Springer-Verlag, Berlin, New York, 1976.

[23] H. J. STETTER, *The defect correction principle and discretization methods*, Numer. Math., 29 (1978), pp. 425-443.

[24] J. VILLADSEN AND M. L. MICHELSEN, *Solution of Differential Equation Models by Polynomial Approximation*, Prentice Hall, Englewood Cliffs, NJ, 1978.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 2, pp. 343-353, March 1990

NODAL SUPERCONVERGENCE AND SOLUTION ENHANCEMENT FOR A CLASS OF FINITE-ELEMENT AND FINITE-DIFFERENCE METHODS*

R. J. MACKINNON† AND G. F. CAREY†

Abstract. A class of finite-element methods for elliptic problems is shown to exhibit nodal superconvergence in the approximate solution, and some equivalence properties to familiar finite-difference operators are demonstrated. The superconvergence property is exploited in a Taylor series analysis to demonstrate Gauss-point superconvergence for the derivatives of the approximation. A post-processing formula for the derivative at the nodes is constructed and shown to exhibit superconvergence. The nodal superconvergence property can be exploited recursively to further enhance the finite-element or finite-difference solution. Supporting numerical studies are given.

Key words. finite element, finite difference, superconvergence, post-processing

AMS(MOS) subject classifications. 65-L60, 65-N30

1. Introduction. In this note we consider a Galerkin finite-element approximation of the Dirichlet problem for the equation $Lu = f$ in Ω. Here Ω is a union of rectangular subdomains, L is a second-order elliptic differential operator with smooth coefficients, and u is assumed to be sufficiently smooth. By introducing an appropriate integration rule for element quadrature we show that the Galerkin approximation u_h, defined on a square mesh of piecewise bilinear elements, is equivalent to a familiar finite-difference approximation of u. Discrete uniform error estimates for this difference approximation are known (Bramble and Hubbard [1]). These estimates imply that difference quotients of the error have the same order of convergence as the error itself; i.e., $O(h^2)$ for the bilinear element. It follows that this nodal superconvergence property holds for the standard Galerkin approximation with higher-order (or full) integration. We use this result to prove new superconvergence results and show how simple and accurate superconvergent post-processing formulas for the solution and derivatives can be derived using Taylor series expansions. Although the formulation and analysis presented here is for problems in two dimensions, the results apply to problems in one dimension as well, and extend directly to three dimensions.

2. Formulation and analysis.

2.1. Nodal solution superconvergence. Consider the boundary value problem

$$(1)\qquad Lu = -\left[\frac{\partial}{\partial x}\left(a_1\frac{\partial u}{\partial x}\right)+\frac{\partial}{\partial y}\left(a_2\frac{\partial u}{\partial y}\right)\right]+b_1u_x+b_2u_y+cu=f$$

in the unit square $\Omega = (0, 1)\times(0, 1)$ with Dirichlet data

$$(2)\qquad u = g \quad \text{on } \partial\Omega.$$

Here we assume that $\mathbf{a}$, $\mathbf{b}$, c, and f are smooth, and L is uniformly elliptic in Ω.

The Galerkin finite-element approximation to (1) is defined to be $u_h \in H^h$, satisfying the essential boundary condition, and such that

$$(3)\qquad B(u_h, v_h) = (f, v_h)$$

* Received by the editors December 16, 1987; accepted for publication (in revised form) June 7, 1988.

† Department of Aerospace Engineering and Engineering Mechanics, University of Texas, Austin, Texas 78712. This research has been supported in part by the Office of Naval Research.

for all $v_h \in H_0^h \subset H_0^1(\Omega)$, where $(\cdot, \cdot)$ is the $L^2(\Omega)$ inner product and $B(\cdot, \cdot)$ is the bilinear functional

$$B(w, \psi) = \int_\Omega [a_1 w_x \psi_x + a_2 w_y \psi_y + (b_1 w_x + b_2 w_y + cw)\psi]\, dx\, dy. \tag{4}$$

Now consider a uniform partition of Ω into square elements of size h and take $H^h(\Omega)$ to be spanned by C^0 piecewise-bilinear functions defined on this partition. Approximating the integrals in (3) by a suitable integration rule applied over each element, we get the approximation $B_h(u_h, v_h) = (f, v_h)_h$ for all $v_h \in H^h$. The resulting algebraic system is

$$\mathbf{B}_h \mathbf{u}_h = \mathbf{f}_h \tag{5}$$

where the precise forms of $\mathbf{B}_h$ and $\mathbf{f}_h$ depend on the particular integration rule used.

For clarity of exposition, let us first consider the case where coefficients $\mathbf{a}$, $\mathbf{b}$, and c are constants, and a (2×2) trapezoidal integration rule is used to evaluate integrals in (3). Accordingly, evaluating the coefficients in (5), for typical interior node point i at (x_i, y_i) with test function v_{hi}, we obtain

$$\begin{aligned} B_h(u_h, v_{hi}) = -\{&a_1[u_h(x_i + h, y_i) - 2u_h(x_i, y_i) + u_h(x_i - h, y_i)] \\ &+ a_2[u_h(x_i, y_i + h) - 2u_h(x_i, y_i) + u_h(x_i, y_i - h)]\} \\ &+ h\frac{b_1}{2}[u_h(x_i + h, y_i) - u_h(x_i - h, y_i)] \\ &+ h\frac{b_2}{2}[u_h(x_i, y_i + h) - u_h(x_i, y_i - h)] + cu_h(x_i, y_i)h^2 \end{aligned} \tag{6}$$

and

$$(f, v_{hi})_h = f(x_i, y_i)h^2. \tag{7}$$

For this case, we see from (6) and (7) that (5) is equivalent to the five-point central difference approximation to (1).

Bramble and Hubbard [1] have shown that, for a solution u of (1) having bounded fifth derivatives, the gridpoint error $e_i = u(x_i, y_i) - u_h(x_i, y_i)$ for the five-point difference approximation satisfies

$$e_i = \phi(x_i, y_i)h^2 + R(x_i, y_i, h)h^3 \tag{8}$$

where ϕ has Lipschitz continuous second derivatives, and R is uniformly bounded in x_i, y_i, and h. It follows from (8) that the solution to this finite-element problem (5) has gridpoint errors of order $O(h^2)$. It should be emphasized that this estimate is a gridpoint result for the discrete problem, and is of the same order as the global L^2 estimate usually encountered in finite-element theory.

Remark. If the domain discretization error is zero (as assumed here), then it follows directly from the proof in Bramble and Hubbard that $\max_{x_i, y_i} |R(x_i, y_i, h)| \leqq Ch$, constant C, so the final term in (8) is actually $O(h^4)$.

In (8) ϕ is the solution to the auxiliary problem

$$\begin{aligned} L\phi &= \tau(u) \quad \text{in } \Omega, \\ \phi &= 0 \qquad\ \ \text{on } \partial\Omega \end{aligned} \tag{9}$$

with τ the truncation error. In particular,

$$B_h(u, v_{hi}) = h^2 Lu(x_i, y_i) + h^4 \tau_i(u) + O(h^6) \tag{10}$$

where $\tau_i(u)$ denotes $\tau(u)$ evaluated at interior gridpoint (x_i, y_i). For trapezoidal integration $\tau(u)$ corresponding to (6) is

$$\tau(u) = -\tfrac{1}{12}[a_1 u_{xxxx} + a_2 u_{yyyy} - 2(b_1 u_{xxx} + b_2 u_{yyy})]. \tag{11}$$

According to (10), the discrete approximation given in (6) and (7) has local truncation errors of order $O(h^4)$. On dividing by h^2, we see that the differential operator is approximated to a local accuracy of order $O(h^2)$. Even if a more accurate quadrature scheme is used for integrating (3), $O(h^4)$ truncation errors remain. Their precise forms depend on the integration rule used. It follows that the estimate in (8) gives the best possible rate for the nodal solution error irrespective of the increase in quadrature accuracy.

This conclusion also holds for the case of smooth variable coefficients, since their variations only introduce $O(h^4)$ truncation errors. (See the Appendix for an example.)

2.2. Derivative superconvergence points. Consider first the problem of derivative calculation from the bilinear finite-element nodal interpolant u_I of u. Simply differentiating the expansion on element Ω_e, we have

$$u_{Ix}(\bar{x}, \bar{y}) = \sum_{j=1}^{4} u_j \psi_{jx}(\bar{x}, \bar{y}) \tag{12}$$

where u_j are the interpolated nodal values for Ω_e, ψ_j are the element basis functions, and $\bar{x}, \bar{y}$ is an arbitrary point in the element.

Next, we introduce Taylor series expansions for $u_j = u(x_j, y_j)$ about $x = \bar{x}$, $y = \bar{y}$ with $\delta_j^x = x_j - \bar{x}$, $\delta_j^y = y_j - \bar{y}$ to obtain

$$\begin{aligned} u_j = u(x_j, y_j) &= u(\bar{x}, \bar{y}) + u_x(\bar{x}, \bar{y})\delta_j^x + u_y(\bar{x}, \bar{y})\delta_j^y \\ &\quad + u_{xx}(\bar{x}, \bar{y})(\delta_j^x)^2/2! + u_{yy}(\bar{x}, \bar{y})(\delta_j^y)^2/2! + u_{xy}(\bar{x}, \bar{y})\delta_j^x\delta_j^y + \cdots. \end{aligned} \tag{13}$$

Using (13) in the right side of (12) and regrouping terms, we have

$$\begin{aligned} \bar{u}_x &= \sum_{j=1}^{4} \bar{\psi}_{jx} u_j - \left[\frac{1}{2!}\sum_{j=1}^{4} \bar{\psi}_{jx}(\delta_j^x)^2\right]\bar{u}_{xx} \\ &\quad - \left[\frac{1}{2!}\sum_{j=1}^{4} \bar{\psi}_{jx}(\delta_j^y)^2\right]\bar{u}_{yy} - \left[\sum_{j=1}^{4} \bar{\psi}_{jx}\delta_j^x\delta_j^y\right]\bar{u}_{xy} + O(h^2) \end{aligned} \tag{14}$$

where for notational convenience $\bar{u} = u(\bar{x}, \bar{y})$, $\bar{u}_x = u_x(\bar{x}, \bar{y})$, $\bar{\psi}_{jx} = \psi_{jx}(\bar{x}, \bar{y})$, and so on. A similar expression holds for $\bar{u}_y$.

Now the derivative of the approximate solution u_h at $\bar{x}, \bar{y}$ in Ω_e is

$$\bar{u}_{hx} = \sum_{j=1}^{4} u_{hj}\bar{\psi}_{jx}. \tag{15}$$

Subtracting (15) from (14) yields the error in the derivative

$$\begin{aligned} \bar{e}_x &= \sum_{j=1}^{4} e_j \bar{\psi}_{jx} - \left[\frac{1}{2!}\sum_{j=1}^{4} \bar{\psi}_{jx}(\delta_j^x)^2\right]\bar{u}_{xx} \\ &\quad - \left[\frac{1}{2!}\sum_{j=1}^{4} \bar{\psi}_{jx}(\delta_j^y)^2\right]\bar{u}_{yy} - \left[\sum_{j=1}^{4} \bar{\psi}_{jx}\delta_j^x\delta_j^y\right]\bar{u}_{xy} + O(h^2). \end{aligned} \tag{16}$$

When we introduce nodal estimate (8) for e_j and use the fact that derivatives and hence difference quotients of ϕ are bounded, the first term on the right in (16) satisfies

$$\left|\sum_{j=1}^{4} \bar{\psi}_{jx} e_j\right| = O(h^2).$$

This implies that (16) will be an $O(h^2)$ approximation, provided the remaining first-order terms are zero or collectively cancel. On examination, we find that coefficients of $\bar{u}_{xy}$ and $\bar{u}_{yy}$ are zero for all $\bar{x}$, $\bar{y}$ in Ω_e, but the coefficient of $\bar{u}_{xx}$ is zero for all $\bar{y}$ with $\bar{x} = (x_1 + x_2)/2$. Therefore, u_{hx} is superconvergent on the *line* bisecting the horizontal sides of Ω_e. Similarly, u_{hy} is $O(h^2)$ along the line bisecting vertical sides of Ω_e. Hence, the centroid (Gauss point) is the superconvergent point for both u_{hx} and u_{hy}.

2.3. Nodal derivative extraction. Now consider the calculation of derivatives (flux components, stresses) at interior node point x_i, y_i. For a solution u of (1) having sufficient smoothness in the interior of Ω, Bramble and Hubbard [1] prove the following estimate for the equivalent finite-difference scheme:

$$(17) \qquad |D_h^n e(x_i, y_i)| \leqq c_n[|e|_{\Omega_h} + O(h^2)]$$

where D_h^n is an nth order difference quotient having $O(h^2)$ truncation error, c_n is a constant independent of h, and $|e|_{\Omega_h} = \max_{x_i, y_i} |e(x_i, y_i)|$. For the problem considered here we have, according to (8),

$$|e|_{\Omega_h} = O(h^2).$$

Thus, (17) becomes

$$(18) \qquad |D_h^n e(x_i, y_i)| \leqq Ch^2.$$

This result can now be used to derive a superconvergent approximation for the flux components $a_1 u_x$, $a_2 u_y$ (and hence derivatives if desired) at node point x_i, y_i.

A Taylor series expansion for $u(x_i - h, y_i)$ about (x_i, y_i) yields

$$(19) \qquad a_1 u_x(x_i, y_i) = \frac{a_1}{h}[u(x_i, y_i) - u(x_i - h, y_i)] + \frac{h}{2} a_1 u_{xx}(x_i, y_i) + O(h^2).$$

Replacing $a_1 u_{xx}(x_i, y_i)$ in (19) using differential equation (1) and then introducing the following difference formulas for a_{1x}, u_x, u_y, and $(a_2 u_y)_y$

$$a_{1x}(x_i, y_i) = \frac{a_1(x_i, y_i) - a_1(x_i - h, y_i)}{h} + O(h),$$

$$u_x(x_i, y_i) = \frac{u(x_i, y_i) - u(x_i - h, y_i)}{h} + O(h),$$

$$u_y(x_i, y_i) = \frac{u(x_i, y_i + h) - u(x_i, y_i - h)}{2h} + O(h^2),$$

$$(a_2 u_y(x_i, y_i))_y = \frac{(a_2(x_i, y_i + h) + a_2(x_i, y_i))}{2h^2}[u(x_i, y_i + h) - u(x_i, y_i)]$$

$$- \frac{(a_2(x_i, y_i) + a_2(x_i, y_i - h))}{2h^2}[u(x_i, y_i) - u(x_i, y_i - h)] + O(h^2),$$

we obtain

$$
\begin{aligned}
a_1u_x(x_i, y_i) = {} & \frac{(a_1(x_i, y_i) + a_1(x_i - h, y_i))}{2h}[u(x_i, y_i) - u(x_i - h, y_i)] \\
& + \frac{(a_2(x_i, y_i + h) + a_2(x_i, y_i))}{4h}[u(x_i, y_i + h) - u(x_i, y_i)] \\
& - \frac{(a_2(x_i, y_i) + a_2(x_i, y_i - h))}{4h}[u(x_i, y_i) - u(x_i, y_i - h)] \\
& + \frac{b_1}{2}[u(x_i, y_i) - u(x_i - h, y_i)] + \frac{b_2}{4}[u(x_i, y_i + h) - u(x_i, y_i - h)] \\
& + c\frac{h}{2}u(x_i, y_i) - \frac{h}{2}f(x_i, y_i) + O(h^2).
\end{aligned}
\tag{20}
$$

Note that (20) is an $O(h^2)$ difference formula involving nodal values of the exact solution u. On introducing the finite-element approximation u_h for u on the right in (20), we define the approximation for $a_1u_x(x_i, y_i)$

$$
\begin{aligned}
a_1u_x^*(x_i, y_i) = {} & \frac{(a_1(x_i, y_i) + a_1(x_i - h, y_i))}{2h}[u_h(x_i, y_i) - u_h(x_i - h, y_i)] \\
& + \frac{(a_2(x_i, y_i + h) + a_2(x_i, y_i))}{4h}[u_h(x_i, y_i + h) - u_h(x_i, y_i)] \\
& - \frac{(a_2(x_i, y_i) + a_2(x_i, y_i - h))}{4h}[u_h(x_i, y_i) - u_h(x_i, y_i - h)] \\
& + \frac{b_1}{2}[u_h(x_i, y_i) - u_h(x_i - h, y_i)] + \frac{b_2}{4}[u_h(x_i, y_i + h) - u_h(x_i, y_i - h)] \\
& + \frac{ch}{2}u_h(x_i, y_i) - \frac{h}{2}f(x_i, y_i).
\end{aligned}
\tag{21}
$$

Subtracting (21) from (20) and using (18), we find that (21) is a superconvergent $O(h^2)$ flux approximation. (For a related study of derivative approximations see MacKinnon and Carey [5].)

Finally, let us use this result to analyze a finite-element projection technique for flux post-processing. This technique is based on the integration-by-parts procedure in the finite-element integral statement, from which we define the projection relationship for $a_1u_x^*$:

$$
\int_{s_i} a_1u_x^*v_{hi}\,ds = \int_{\Omega_p} (a_1u_{hx}v_{hix} + a_2u_{hy}v_{hiy} + (\mathbf{b}\cdot\nabla u_h + cu_h - f)v_{hi})\,dx\,dy \tag{22}
$$

where s_i is defined by element sides connecting gridpoints $(x_i, y_i - h)$, (x_i, y_i), $(x_i, y_i + h)$ and Ω_p represents the two-element patch defined by gridpoints $(x_i - h, y_i)$, (x_i, y_i), $(x_i, y_i + h)$, $(x_i - h, y_i + h)$, $(x_i, y_i - h)$, $(x_i - h, y_i - h)$. This approach has been examined in one dimension by Wheeler [6], Dupont [4], and Carey [2]. In two-dimensional numerical test cases the method has been demonstrated to yield an $O(h^2)$ approximation to the nodal flux when $a_1u_x^*$ is assumed to be piecewise-constant over s_i (Carey, Chow, and Seager [3]). Indeed, if we integrate (21) using the trapezoidal rule, as described in the Appendix, then the resulting discrete formula for $a_1u_x^*$ is identical to (21). This then confirms the observed numerical convergence rate of $O(h^2)$. If a more accurate quadrature scheme is used to evaluate (22), the resulting difference formula is, in general, different from (21). However, a simple Taylor series analysis confirms that the formula is $O(h^2)$ accurate.

3. Nodal solution enhancement. In this section we apply the results obtained in the foregoing analysis and formulate a new scheme to compute an accurate approximation e_i^* to the gridpoint error $e_i = u(x_i, y_i) - u_h(x_i, y_i)$. This approximation may then be used to improve u_h and, moreover, increase the asymptotic rate of convergence of u_h and its derivatives. We point out that although the formulation presented here is for problems in two-dimensions, it includes the one-dimensional case by simply setting y derivatives equal to zero.

In the interest of clarity, we restrict our analysis to the constant coefficient case described by (6)-(11). The extension to other cases involving different quadrature schemes and variable coefficients is straightforward in view of our previous results.

First recall (10) and (11):

$$B_h(u, v_{hi}) = h^2 Lu(x_i, y_i) + h^4 \tau_i(u) + O(h^6), \tag{10}$$

$$\tau(u) = -\tfrac{1}{12}[a_1 u_{xxxx} + a_2 u_{yyyy} - 2(b_1 u_{xxx} + b_2 u_{yyy})]. \tag{11}$$

Now from estimate (18), since u_h is $O(h^2)$ accurate at the node points, any nth-order difference quotient D_h^n of u_h also converges to the exact value $D^n u$ at a rate of $O(h^2)$. Therefore, at any interior node point i, $\tau(u)$ can be rewritten using difference quotients $D_h^n u_h$ as

$$\begin{aligned}\tau_i(u) &= -\tfrac{1}{12}[a_1 D_{hxi}^4 u_h + a_2 D_{hyi}^4 u_h - 2(b_1 D_{hxi}^3 u_h + b_2 D_{hyi}^3 u_h)] + O(h^2)\\ &= \tau_{hi}(u_h) + O(h^2).\end{aligned} \tag{23}$$

(Note that since the fifth derivatives of u are assumed bounded, then $\tau_i(u)$ at node points on boundary $\partial\Omega$ can also be approximated to $O(h^2)$ accuracy by simply using an $O(h^2)$ extrapolation to the boundary.)

Next, interpolate the nodal values $\tau_i(u)$ in the piecewise-bilinear basis as

$$\tau(u) = \sum_{j=1}^{N} \tau_j(u)\psi_j(x, y) + O(h^2) \tag{24}$$

where N is the number of node points.

Introducing (23) in (24), we have

$$\tau(u) = \sum_{j=1}^{N} \tau_{hj}(u_h)\psi_j(x, y) + O(h^2). \tag{25}$$

Replacing $\tau_i(u)$ in (10) using (25), we have

$$B_h(u, v_{hi}) = h^2 Lu(x_i, y_i) + h^4 \tau_{hi}(u_h) + O(h^6). \tag{26}$$

Using (26) in place of (10), the estimate (8) now has the form

$$e_i = \phi^*(x_i, y_i)h^2 + R^*(x_i, y_i, h)h^3 \tag{27}$$

where ϕ^* satisfies the auxiliary problem

$$\begin{aligned}L\phi^* &= \tau_h(u_h) = \sum_{j=1}^{N} \tau_{hj}(u_h)\psi_j(x, y) \quad \text{in } \Omega,\\ \phi^* &= 0 \quad \text{on } \partial\Omega.\end{aligned} \tag{28}$$

The objective now is to construct a finite-element approximation ϕ_h^* to ϕ^* in (28), and then use this approximation in the leading term on the right of (28) to obtain an accurate correction to the nodal solution.

First, let us assume that we have already computed u_h from (5) using LU factorization and have saved the computed matrix factors. The Galerkin finite-element approximation to (28) is as follows. Find $\phi_h^* \in H^h$ such that

$$B(\phi_h^*, v_h) = (\tau_h(u_h), v_h) \tag{29}$$

for all $v_h \in H^h \subset H_0^1$. As before, if we evaluate integrals in (29) using the (2×2) trapezoidal integration rule we get

$$\mathbf{B}_h \boldsymbol{\phi}_h^* = \boldsymbol{\tau}_h(u_h) h^2 \tag{30}$$

where $\tau_{hi}(u_h)$ is defined in (23) and $B_h(\phi_h^*, v_{hi})$ is analogous to (6).

Since the matrix factorization of $\mathbf{B}_h$ is already given from the previous calculation of $\mathbf{u}_h$, the approximate function ϕ_h^* in (30) can be computed efficiently once $\tau_{hi}(u_h)$ are computed (for the Dirichlet problem $\tau_{hi}(u_h)$ is needed at interior points only); $\tau_{hi}(u_h)$ is easily computed using one-dimensional difference formulas. For example, we may write

$$\begin{aligned} D_\xi^n u &= \sum_{j=1}^{k+1} \frac{d^n}{d\xi^n} \Psi_j(\xi) u(\xi_j) + O(h^{k+1-n}) \\ &= D_{h\xi}^n u + O(h^{k+1-n}), \qquad k \geqq n, \quad \xi = x, y \end{aligned} \tag{31}$$

where $u(\xi_j)$ are node point values of function u, and Ψ_j are Lagrange polynomial shape functions of degree k. In particular, for a second-order $(O(h^2))$ approximation to d^4u/dx^4, $n=4$ and $k=5$. Note that (31) can be used to approximate d^4u/dx^4 at interior nodes near the boundary. For this case (31) is simply a one-sided difference formula involving interior node point values of u only.

Solution ϕ_h^* from (30) will approximate ϕ^* with accuracy $O(h^p)$ at all node points, where p depends on the smoothness of solution u to (1). Note that τ in (24) is C^1 in view of the assumptions on u. Moreover, τ_h in (28) is C^0 by construction so $\phi^* \in C^2$ and $p \geqq 1$. Replacing ϕ^* in (28) with ϕ_h^*, we have

$$\begin{aligned} e_i^* &= \phi_h^*(x_i, y_i) h^2 + R^*(x_i, y_i, h) h^3 + O(h^{2+p}) \\ &= \phi_{hi}^* h^2 + O(h^{2+p}) \end{aligned} \tag{32}$$

since R^* is $O(h)$.

This important result implies that we can compute node point errors e_i^* having at least $O(h^3)$ accuracy, and $O(h^4)$ accuracy $(p=2)$ for sufficiently smooth solution. An immediate consequence of this result is that we can also increase the accuracy of our approximation u_h (and its derivatives, if desired) from $O(h^2)$ to at least $O(h^3)$. That is, the enhanced gridpoint value obtained by adding the nodal correction e_i^* becomes

$$u_{hi}^* = u_{hi} + e_i^*. \tag{33}$$

The solution enhancement procedure may be summarized as follows:

(1) Solve the finite-element problem $\mathbf{B}_h \mathbf{u}_h = \mathbf{f}_h$ using sparse LU factorization and save matrix factors.

(2) "Process" approximation $\mathbf{u}_h$ and form associated vector $\boldsymbol{\tau}_h(u_h)$. Then, using matrix factors of $\mathbf{B}_h$, solve auxiliary problem $\mathbf{B}_h \boldsymbol{\phi}_h^* = \boldsymbol{\tau}_h(u_h)$.

(3) Compute approximate node point error correction

$$e_i^* = \phi_h^*(x_i, y_i)h^2$$

and hence the "enhanced" solution

$$u_i^* = u_{hi} + e_i^*.$$

3.1. Numerical examples. Numerical test studies have been made to demonstrate the effectiveness of the nodal enhancement post-processing procedure. (For results related to the application of post-processing derivative formula (22) and related formulas, we refer the reader to Carey [2], Carey, Chow, and Seager [3], and MacKinnon and Carey [5].)

In the first test case we consider the two-point boundary value problem

$$\begin{aligned} -u_{xx} + u_x + u &= f, \qquad 0 < x < 1, \\ u(0) = u(1) &= 0 \end{aligned} \tag{34}$$

where f is constructed such that the analytic solution is $u = x(1-x)(1+x)^5$.

We take a sequence of uniform mesh refinements with $h = \frac{1}{5}, \frac{1}{10}, \frac{1}{20}$, and $\frac{1}{40}$. Numerical integration is performed using the trapezoidal rule, and derivatives u_{xxx} and u_{xxxx} in τ are approximated to order $O(h^3)$ and $O(h^2)$, respectively, by six-point difference formulas. A six-point formula for u_{xxx} was used because it is computationally convenient to simply differentiate this formula and use the result to approximate u_{xxxx}.

Node point errors E_i, E_i^* for approximations u_{hi} and enhancement u_i^* are presented in Table 1. Note the substantial increase in accuracy and asymptotic rates of convergence afforded by the enhancement procedure.

Next we examine three approximations to u_x at $x = 1$. These approximations are: the standard $O(h)$ derivative approximation u_{hx}; the post-processed derivative u_x^* given by (21); and the enhanced derivative denoted by u_x^{**} and given by an $O(h^4)$ six-point difference formula operating on enhanced solution u^*. Results are presented in Table 2. Approximations u_x^* and u_x^{**} are $O(h^2)$ and $O(h^4)$ accurate as predicted.

TABLE 1
Node point errors $E(x_i)$, $E^*(x_i)$ for the case $b = 1$.

h	$E(0.2)$	$E^*(0.2)$	$E(0.4)$	$E^*(0.4)$	$E(0.6)$	$E^*(0.6)$	$E(0.8)$	$E^*(0.8)$
$\frac{1}{5}$	0.085004	0.016524	0.161265	0.010216	0.200329	0.009952	0.162612	0.015906
$\frac{1}{10}$	0.021283	0.204E−3	0.040342	0.598E−3	0.050081	0.724E−3	0.040634	0.389E−3
$\frac{1}{20}$	0.005323	0.27E−4	0.010087	0.51E−4	0.012520	0.61E−4	0.010157	0.46E−4
$\frac{1}{40}$	0.001330	0.18E−5	0.002521	0.33E−5	0.0031301	0.40E−5	0.002539	0.31E−5
	$\sim O(h^2)$	$\sim O(h^4)$	$\sim O(h^2)$	$\sim O(h^4)$	$\sim O(h^2)$	$\sim O(h^4)$	$\sim O(h^2)$	$\sim O(h^4)$

TABLE 2
Derivatives u_{hx}, u_x^*, and u_x^{**} at $x = 1$. The exact derivative is $u_x(1) = -32.0$.

h	u_{hx}	u_x^*	u_x^{**}	$\lvert u_x - u_{hx}\rvert$	$\lvert u_x - u_x^*\rvert$	$\lvert u_x - u_x^{**}\rvert$
$\frac{1}{5}$	−14.303480	−34.933828	−31.476970	17.69652	2.933828	0.5230300
$\frac{1}{10}$	−22.034701	−32.736437	−31.982434	9.965299	0.7364370	0.0175660
$\frac{1}{20}$	−26.716385	−32.184295	−31.999583	5.283615	0.1842950	0.417E−3
$\frac{1}{40}$	−29.280084	−32.046085	−31.999975	2.719916	0.0460850	0.25E−4
				$\sim O(h)$	$\sim O(h^2)$	$\sim O(h^4)$

As a two-dimensional test problem we take the example

$$(u_{xx}+u_{yy})=(2-42x^5)(y-y^7)-42y^5(x^2-x^7) \qquad \text{in } \Omega=(0,1)\times(0,1)$$

with

$$u=0 \quad \text{on } \partial\Omega. \tag{35}$$

The analytic solution is the polynomial

$$u=(x^2-x^7)(y-y^7). \tag{36}$$

Node point results for a sequence of calculations on uniformly refined meshes of $h=\frac{1}{5},\frac{1}{10}$, and $\frac{1}{30}$ are given in Tables 3-5. Again, the observed rates of convergence corroborate our analysis.

Conclusion. By a suitable choice of quadrature rule the finite-element approximation for a two-dimensional elliptic problem has been related to a familiar finite-difference approximation. Nodal superconvergence of the solution then follows from an estimate of finite-difference theory. Moreover, any nth order difference approximation having Taylor series truncation error of $O(h^2)$ at a node point converges to the exact value at a rate of $O(h^2)$. Therefore, accurate derivative extraction formulas can be derived directly using Taylor series ideas.

TABLE 3
Node point errors $E(x_i)$, $E^*(x_i)$ along $x=0.8$.

h	$E(0.8,0.2)$	$E^*(0.8,0.2)$	$E(0.8,0.4)$	$E^*(0.8,0.4)$	$E(0.8,0.6)$	$E^*(0.8,0.6)$	$E(0.8,0.8)$	$E^*(0.8,0.8)$
$\frac{1}{5}$	0.012138	0.004074	0.023837	0.005127	0.032477	0.006333	0.030038	0.006824
$\frac{1}{10}$	0.003102	0.40E−4	0.006087	0.60E−4	0.008275	0.80E−4	0.007625	0.93E−4
$\frac{1}{30}$	0.346E−3	0.2E−6	0.680E−3	0.3E−6	0.924E−3	0.4E−6	0.850E−3	0.4E−6
	$\sim O(h^2)$	$\sim O(h^4)$	$\sim O(h^2)$	$\sim O(h^4)$	$\sim O(h^2)$	$\sim O(h^4)$	$\sim O(h^2)$	$\sim O(h^4)$

TABLE 4
Node point errors $E(x_i)$, $E^*(x_i)$ along $y=0.8$.

h	$E(0.2,0.8)$	$E^*(0.2,0.8)$	$E(0.4,0.8)$	$E^*(0.4,0.8)$	$E(0.6,0.8)$	$E^*(0.6,0.8)$
$\frac{1}{5}$	0.007514	0.003668	0.017644	0.003680	0.028428	0.005004
$\frac{1}{10}$	0.001982	0.21E−4	0.004496	0.22E−4	0.007233	0.48E−4
$\frac{1}{30}$	0.214E−3	0.2E−6	0.502E−3	0.4E−6	0.807E−3	0.5E−6
	$\sim O(h^2)$	$\sim O(h^4)$	$\sim O(h^2)$	$\sim O(h^4)$	$\sim O(h^2)$	$\sim O(h^4)$

TABLE 5
Derivatives u_{hx}, u_x^*, and u_x^{**} at $(x,y)=(0.8,0.8)$. The exact derivative is $u_x(0.8,0.8)=-0.1387215$.

h	u_{hx}	u_x^*	u_x^{**}	$\lvert u_x-u_{hx}\rvert$	$\lvert u_x-u_x^*\rvert$	$\lvert u_x-u_x^{**}\rvert$
$\frac{1}{5}$	0.282010	−0.405757	−0.155165	0.420731	0.267035	0.016443
$\frac{1}{10}$	0.137307	−0.208948	−0.139683	0.276028	0.070227	0.961E−3
$\frac{1}{30}$	−0.0310355	−0.146723	−0.138726	0.107686	0.008001	0.4E−5
				$\sim O(h)$	$\sim O(h^2)$	$\sim O(h^4)$

We emphasize that since this Taylor series approach relies only on elementary analysis concepts, it is straightforward to understand and implement. Furthermore, although this is not taken up here, the method can be easily applied to higher-order elements and problems in three dimensions. Also, derivatives can be extracted from finite-difference solutions in the same manner.

Finally, using the truncation error in an auxiliary problem the nodal superconvergence property can be further exploited to enhance the gridpoint solution accuracy. These results are of practical significance in solution and derivative post-processing and also for a posteriori error analysis in conjunction with adaptive refinement. The adaptive refinement aspects will be taken up in future studies.

Appendix. Trapezoidal rule and variable coefficients. For the case of variable coefficients and trapezoidal integration, we have from (3) at interior gridpoint i

$$B_h(u_h, v_{hi}) = f_{hi} \tag{1.1}$$

where

$$\begin{aligned} B_h(u_h, v_{hi}) = -\bigg\{ & \frac{a_1(x_i+h, y_i)+a_1(x_i, y_i)}{2h}[u_h(x_i+h, y_i)-u_h(x_i, y_i)] \\ & -\frac{a_1(x_i, y_i)+a_1(x_i-h, y_i)}{2h}[u_h(x_i, y_i)-u_h(x_i-h, y_i)] \\ & +\frac{a_2(x_i, y_i+h)+a_2(x_i, y_i)}{2h}[u_h(x_i, y_i+h)-u_h(x_i, y_i)] \\ & -\frac{a_2(x_i, y_i)+a_2(x_i, y_i-h)}{2h}[u_h(x_i, y_i)-u_h(x_i, y_i-h)]\bigg\} \\ & +h\frac{b_1}{2}(x_i, y_i)[u_h(x_i+h, y_i)-u_h(x_i-h, y_i)] \\ & +h\frac{b_2}{2}(x_i, y_i)[u_h(x_i, y_i+h)-u_h(x_i, y_i-h)]+c(x_i, y_i)u_h(x_i, y_i)h^2 \end{aligned} \tag{1.2}$$

and

$$f_{hi} = f(x_i, y_i)h^2. \tag{1.3}$$

For a smooth function w, and using Taylor's theorem,

$$B_h(w, v_{hi}) = h^2Lw(x_i, y_i)+h^4\tau_i(w)+O(h^6) \tag{1.4}$$

where

$$\begin{aligned} \tau(w) = -\tfrac{1}{12}[& a_1w_{xxxx}+a_{1x}w_{xxx}+a_{1xx}w_{xx}+a_{1xxx}w_x \\ & +a_2w_{yyyy}+a_{2y}w_{yyy}+a_{2yy}w_{yy}+a_{2yyy}w_y-2(b_1w_{xxx}+b_2w_{yyy})]. \end{aligned} \tag{1.5}$$

From (1.4) and (1.5) we see that variable coefficients produce additional $O(h^4)$ terms in τ. Hence, the accuracy of (1.1) remains $O(h^2)$ and the results demonstrated for the constant coefficient case extend directly to the case of variable coefficients.

REFERENCES

[1] J. H. BRAMBLE AND B. E. HUBBARD, *Approximation of derivatives by finite difference methods in elliptic boundary value problems*, Contrib. Differential Equations, 3 (1964), pp. 399-410.

[2] G. F. CAREY, *Derivative calculation from finite element solutions*, Comput. Methods Appl. Mech. Engrg., 35 (1982), pp. 1-14.

[3] G. F. CAREY, S. S. CHOW, AND M. SEAGER, *Approximate boundary flux calculation*, J. Comput. Meth. Appl. Mech. Angrg., 50 (1985), pp. 107-120.

[4] T. DUPONT, *A unified theory of superconvergence for Galerkin methods for two-point boundary problems*, SIAM J. Numer. Anal., 13 (1976), pp. 362-368.

[5] R. J. MACKINNON AND G. F. CAREY, *Analysis of material interface discontinuities and superconvergent fluxes in finite difference theory*, J. Comput. Phys., 75 (1988), pp. 151-167.

[6] M. F. WHEELER, *A Galerkin procedure for estimating the flux for two-point problems*, SIAM J. Numer. Anal., 11 (1974), pp. 764-768.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 2, pp. 354-367, March 1990

VECTORIZATION OF THE ODD-EVEN HOPSCOTCH SCHEME AND THE ALTERNATING DIRECTION IMPLICIT SCHEME FOR THE TWO-DIMENSIONAL BURGERS EQUATIONS*

E. D. DE GOEDE† AND J. H. M. TEN THIJE BOONKKAMP‡

Abstract. A vectorized version of the odd-even hopscotch (OEH) scheme and the alternation direction implicit (ADI) scheme have been implemented on vector computers for solving the two-dimensional Burgers equations on a rectangular domain. This paper examines the efficiency of both schemes on vector computers. Data structures and techniques employed in vectorizing both schemes are discussed, accompanied by performance details.

Key words. vector computers, Burgers equations, odd-even hopscotch scheme, alternating direction implicit scheme, vectorization

AMS(MOS) subject classifications. primary 65V05; secondary 65M05, 76DXX

1. Introduction. This report is written as a contribution to a project for developing numerical software for vector- and parallel computers. Vectorized versions of the odd-even hopscotch (OEH) scheme and the alternating direction implicit (ADI) scheme are developed in FORTRAN77 for the two-dimensional Burgers equations. In the near future, the vectorized codes will be combined with a pressure correction technique [8], [13] in order to solve the time-dependent, incompressible, Navier-Stokes equations.

The OEH scheme and the ADI scheme are integration schemes for time-dependent partial differential equations (PDEs) and are applicable to wide classes of problems. The OEH scheme has shown to be an efficient scheme on serial (scalar) computers, in the sense that it is fast per timestep. Moreover, the scheme is relatively easy to implement. Due to its near-explicitness the OEH scheme is also very suitable for use on vector computers. A detailed discussion of the OEH scheme is given in [4]. The ADI scheme we consider in this report is the Peaceman-Rachford scheme [11]. The ADI scheme is more expensive per timestep than the OEH scheme since it requires the solution of tridiagonal systems of equations. However, the ADI scheme is more robust than the OEH scheme.

For the solution of the tridiagonal systems we use the Gaussian elimination method, a variant of the partition method of Wang [17], which is described in [3], [9], and a method developed by De Goede and Wubs [3]. By the approach of De Goede and Wubs, the tridiagonal systems are solved by a combination of explicit and implicit calculations, thus resulting in an alternating direction explicit-implicit (ADEI) scheme. Since the Gaussian elimination method is a sequential method, this method seems to be unsuitable for use on vector computers. However, for the two-dimensional ADI scheme a number of (independent) tridiagonal systems must be solved. Therefore, this method allows vectorization across the systems. Moreover, this method does not increase the operation count, unlike the above-mentioned partition methods. Furthermore, it turns out that also the partition method and the explicit-implicit method are efficient on vector computers.

* Received by the editors November 18, 1987; accepted for publication (in revised form) November 30, 1988.

† Centre for Mathematics and Computer Science, P.O. Box 4079, 1009 AB Amsterdam, the Netherlands.

‡ Nederlandse Philips Bedryven B.V., P.O. Box 80000, 5600 JA Eindhoven, the Netherlands.

The purpose of this paper is to report our experience in vectorizing both schemes for the two-dimensional Burgers equations. Much effort has been spent in optimizing the FORTRAN code for vector computers, avoiding the explicit use of assembler code. The experiments have been carried out on a (2-pipe) CDC Cyber 205 and a Cray X-MP/24. We used one (portable) code on both machines. Since the code contains many long vector operations, it is our opinion that on other vector machines (such as the NEC SX-2, Fujitsu VP200, Alliant FX/8, etc.) we will also obtain good performances.

Section 2 contains a brief summary of the conceptual features of vector computers, which are relevant to the present application. In § 3 a description of the OEH scheme and the ADI scheme is given. Section 4 is devoted to the description of the techniques used for vectorizing both the OEH scheme and the ADI scheme. In § 5, we compare the accuracy and performance of both schemes. Finally, § 6 contains some concluding remarks.

2. Vector processing. Vector operations fall into two main categories: those that perform floating-point arithmetic, and those that may be called data-motion operations (for example, operations to compress or expand an array using an index-list). The need for vector data-motion operations also becomes apparent when we consider the definition of a vector on a CDC Cyber 205: a vector is a set of similar elements occupying consecutive memory locations. The reason for this vector definition is that when performing vector operations on a CDC Cyber 205 the input elements stream directly from the memory to the vector pipes (arithmetic units) and the output elements stream directly back into the memory. A Cray-computer accepts vectors for which the number of memory locations between consecutive elements (the so-called stride) is constant.

To enhance an effective data flow rate in order to match the computation rate of vector computers, the memory is divided into memory banks that may operate concurrently. For example, the memory of the CDC Cyber 205 is divided into 16 memory stacks, each of which is divided into eight independent banks. When one memory stack is busy with a memory request, further references to the same stack cannot be made. If a vector operation calls for an operand whose elements are located w words apart in the memory (i.e., stride w), then the data flow rate might be reduced due to the memory conflicts and thus result in a longer vector operation time. So, in order to obtain a good performance on vector computers it is important to consider the data structure very carefully (see § 4) [6].

For an efficient use of vector computers, the compiler plays an important role. The compiler translates FORTRAN DO-loops into vector machine instructions, if possible. This process is called auto-vectorization. The nature of vector operations is such that only DO-loops are candidates for vector operations. Specific characteristics of a given DO-loop determine its vectorizability [1]. It is not always possible to vectorize a code, as in the following example:

$$\begin{gathered}\text{DO } 10\ I=1,\ N\\ A(I+1)=A(I)+S \qquad (2.1)\\ 10\ \text{CONTINUE}\end{gathered}$$

Because in vector processing the arguments must be determinable before the operation starts, this loop cannot be vectorized. This restriction is known as recursion; it conflicts with the nature of vector processing.

In many situations the compiler can be instructed to generate more efficient codes. We have used such instructions, e.g., in the following situation. The compiler can be instructed to vectorize DO-loops, ignoring possible vector dependencies, by inserting a so-called comment-directive:

for the CFT77 compiler (Cray X-MP/24):	CDIR$ IVDEP
and for the VAST compiler (Cyber 205):	CVD$ NODEPCHK

3. The OEH scheme and the ADI scheme for the two-dimensional Burgers equations. Consider the two-dimensional Burgers equations:

$$\begin{aligned} u_t &= f_1(u, v) \quad \text{with } f_1(u, v) = -uu_x - vu_y + (u_{xx} + u_{yy})/\text{Re}, \\ v_t &= f_2(u, v) \quad \text{with } f_2(u, v) = -uv_x - vv_y + (v_{xx} + v_{yy})/\text{Re}, \end{aligned} \tag{3.1}$$

with Re denoting the Reynolds number and u and v the velocity components in x- and y-direction, respectively. On the boundary Γ of the connected space domain Ω, we prescribe the Dirichlet conditions

$$u = u_\Gamma, \qquad v = v_\Gamma.$$

The Burgers equations have the same convective and viscous terms as the incompressible Navier–Stokes equations, although the pressure gradient terms are not retained. Also a solution to the Burgers equations would not, in general, satisfy the continuity equation. These equations possess the desirable property that exact solutions can be constructed by means of the Cole–Hopf transformation [2]. This enables us to compare the numerical solution of the Burgers equations with the exact solution.

In this section we give a description of the OEH scheme and the ADI scheme for the Burgers equations. The space discretization is discussed in § 3.1 and the time integration in § 3.2.

3.1. Space discretization. For the space discretization the computational domain is covered by a $N \times M$ rectangular staggered grid, with h and k being the grid sizes in x- and y-direction, respectively (see Fig. 1). In a staggered grid different variables are defined at different grid points. The reason for this choice is that in continuation to this report we want to apply the OEH scheme and the ADI scheme to the incompressible Navier–Stokes equations, for which a staggered grid is most suitable [14].

In what follows, U is a grid function approximating the velocity u (likewise for V, F_1, and F_2) with components U_{ij}. The components U_{ij} are numbered lexicographically. The application of standard second-order central differences converts (3.1) into

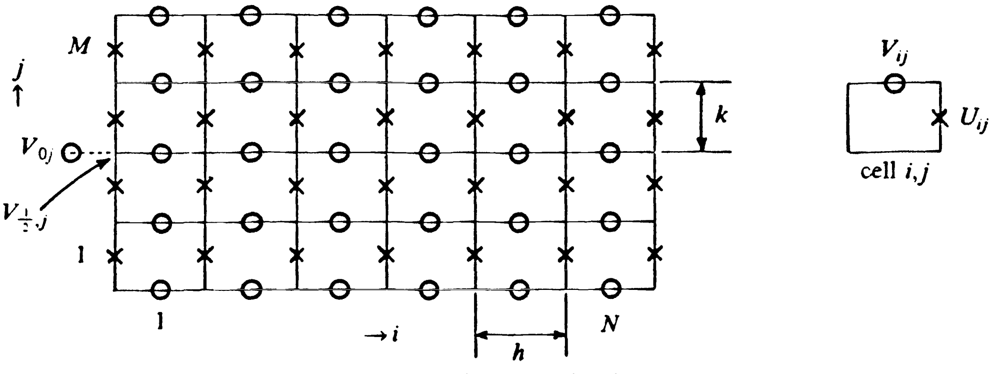


FIG. 1. *The staggered grid.*

the system of ordinary differential equations (ODEs):

(3.2a) $\frac{d}{dt} U_{ij} = F_{1,ij}(U, V),\ i=1,\cdots,N-1,\ j=1,\cdots,M$ (interior ×-points),

(3.2b) $\frac{d}{dt} V_{ij} = F_{2,ij}(U, V),\ i=1,\cdots,N,\quad j=1,\cdots,M-1$ (interior O-points),

where

(3.3a)
$$F_{1,ij} = -U_{ij}\frac{(U_{i+1,j}-U_{i-1,j})}{2h} - \bar{V}_{ij}\frac{(U_{i,j+1}-U_{i,j-1})}{2k} + \frac{1}{\text{Re}}\frac{(U_{i+1,j}-2U_{ij}+U_{i-1,j})}{h^2} + \frac{1}{\text{Re}}\frac{(U_{i,j+1}-2U_{ij}+U_{i,j-1})}{k^2},$$

(3.3b)
$$F_{2,ij} = -\bar{U}_{ij}\frac{(V_{i+1,j}-V_{i-1,j})}{2h} - V_{ij}\frac{(V_{i,j+1}-V_{i,j-1})}{2k} + \frac{1}{\text{Re}}\frac{(V_{i+1,j}-2V_{ij}+V_{i-1,j})}{h^2} + \frac{1}{\text{Re}}\frac{(V_{i,j+1}-2V_{ij}+V_{i,j-1})}{k^2}.$$

In (3.3a) $\bar{V}_{ij}$ represents an approximation to V at the ×-points; likewise in (3.3b) $\bar{U}_{ij}$ represents an approximation to U at the O-points. The values of $\bar{U}_{ij}$ and $\bar{V}_{ij}$ are determined by averaging over neighbouring values. For the ADI scheme $\bar{U}_{ij}$ and $\bar{V}_{ij}$ are trivially defined by

(3.4a) $\bar{U}_{ij} = \frac{1}{4}(U_{ij}+U_{i,j+1}+U_{i-1,j}+U_{i-1,j+1}),\qquad \bar{V}_{ij} = \frac{1}{4}(V_{ij}+V_{i,j-1}+V_{i+1,j}+V_{i+1,j-1}).$

However, for the OEH scheme we choose

(3.4b) $\bar{U}_{ij} = \frac{1}{2}(U_{i-1,j}+U_{i,j+1}),\qquad \bar{V}_{ij} = \frac{1}{2}(V_{i,j-1}+V_{i+1,j}).$

The reason for this choice will become apparent in § 3.2.1.

For the treatment of the boundary conditions, we apply a simple reflection technique [13]. Consider, e.g., (3.2b) and (3.3b) at the O-points $(1, j)$, which involves the outside value $V_{0,j}$. The reflection technique consists of writing the boundary value $V_{1/2,j}$ as a mean value of its neighbouring values $V_{0,j}$ and $V_{1,j}$, so that $V_{0,j} = 2V_{1/2,j} - V_{1,j}$ (see Fig. 1).

3.2. Time integration. Let $\mathbf{U} = (U, V)^T$ and $\mathbf{F}(\mathbf{U}) = (F_1(U, V), F_2(U, V))^T$; then (3.2) can be written in the vector form

(3.5)
$$\frac{d}{dt}\mathbf{U} = \mathbf{F}(\mathbf{U}).$$

For reasons of computational feasibility, we apply a two-term splitting formula for the numerical integration of (3.5). Let

$$\mathbf{F}(\mathbf{U}) = \mathbf{F}_1(\mathbf{U}) + \mathbf{F}_2(\mathbf{U}),$$

and consider the two-stage formula

(3.6)
$$\mathbf{U}^{n+1/2} = \mathbf{U}^n + \tfrac{1}{2}\tau[\mathbf{F}_1(\mathbf{U}^{n+1/2}) + \mathbf{F}_2(\mathbf{U}^n)],$$
$$\mathbf{U}^{n+1} = \mathbf{U}^{n+1/2} + \tfrac{1}{2}\tau[\mathbf{F}_1(\mathbf{U}^{n+1/2}) + \mathbf{F}_2(\mathbf{U}^{n+1})],$$

with τ denoting the timestep. It can be easily verified that this integration formula is second-order consistent for any ODE system (3.5) [7]. Both the OEH scheme and the ADI scheme are special cases of (3.6).

3.2.1. The OEH scheme. In what follows, U^n_{ij} denotes the discrete approximation to u at the grid point (ih, jk) at time level $t_n = n\tau$ (likewise for V^n_{ij}). The OEH scheme for (3.5) is given by the numerical integration formula [4]

$$U^{n+1}_{ij} - \tau\theta^{n+1}_{ij} F_{ij}(\mathbf{U}^{n+1}) = U^n_{ij} + \tau\theta^n_{ij} F_{ij}(\mathbf{U}^n), \tag{3.7}$$

where $U^n_{ij} = (U^n_{ij}, V^n_{ij})^T$ (likewise for $F_{ij}(\mathbf{U}^n)$). The function θ^n_{ij} is defined by

$$\theta^n_{ij} = \begin{cases} 1 & \text{if } n+i+j \text{ is odd} \quad \text{(odd points)}, \\ 0 & \text{if } n+i+j \text{ is even} \quad \text{(even points)}. \end{cases} \tag{3.8}$$

Writing down two successive steps of (3.7) yields

$$U^{n+1}_{ij} = U^n_{ij} + \tau\theta^n_{ij} F_{ij}(\mathbf{U}^n) + \tau\theta^{n+1}_{ij} F_{ij}(\mathbf{U}^{n+1}), \tag{3.9a}$$

$$U^{n+2}_{ij} = U^{n+1}_{ij} + \tau\theta^{n+1}_{ij} F_{ij}(\mathbf{U}^{n+1}) + \tau\theta^{n+2}_{ij} F_{ij}(\mathbf{U}^{n+2}). \tag{3.9b}$$

Let $\mathbf{F}_O(\mathbf{U})$ and $\mathbf{F}_E(\mathbf{U})$ denote the restriction of $\mathbf{F}(\mathbf{U})$ to the odd and even points respectively, then replacing τ by $\tau/2$, (3.9) can be written in the form

$$\mathbf{U}^{n+1/2} = \mathbf{U}^n + \tfrac{1}{2}\tau[\mathbf{F}_E(\mathbf{U}^{n+1/2}) + \mathbf{F}_O(\mathbf{U}^n)], \tag{3.10a}$$

$$\mathbf{U}^{n+1} = \mathbf{U}^{n+1/2} + \tfrac{1}{2}\tau[\mathbf{F}_E(\mathbf{U}^{n+1/2}) + \mathbf{F}_O(\mathbf{U}^{n+1})]. \tag{3.10b}$$

The order of computation for the OEH scheme is

$$\mathbf{U}^{n+1/2}_O = \mathbf{U}^n_O + \tfrac{1}{2}\tau\mathbf{F}_O(\mathbf{U}^n) \qquad (= 2\mathbf{U}^n_O - \mathbf{U}^{n-1/2}_O \text{ if } n \geqq 1), \tag{3.11a}$$

$$\mathbf{U}^{n+1/2}_E = \mathbf{U}^n_E + \tfrac{1}{2}\tau\mathbf{F}_E(\mathbf{U}^{n+1/2}), \tag{3.11b}$$

$$\mathbf{U}^{n+1}_E = \mathbf{U}^{n+1/2}_E + \tfrac{1}{2}\tau\mathbf{F}_E(\mathbf{U}^{n+1/2}) \qquad (= 2\mathbf{U}^{n+1/2}_E - \mathbf{U}^n_E), \tag{3.11c}$$

$$\mathbf{U}^{n+1}_O = \mathbf{U}^{n+1/2}_O + \tfrac{1}{2}\tau\mathbf{F}_O(\mathbf{U}^{n+1}). \tag{3.11d}$$

Note that (3.11a) is just the forward Euler rule at the odd points, whereas (3.11b) is the backward Euler rule at the even points. For (3.11c) and (3.11d) it is just vice versa. Substituting (3.4b) into (3.3), it can be easily verified that in (3.11) there exists an odd–even coupling between the variables, i.e., a variable at an odd point is only coupled to variables at even points and vice versa. Because of this odd–even coupling and the alternating use of the forward- and backward Euler rule, scheme (3.11) is only diagonally implicit. Note that the computation of the forward Euler rule in (3.11a) and (3.11c) can be economized by using a simple interpolation formula. The scheme thus obtained is called the fast form of the OEH scheme [4].

3.2.2. The ADI scheme. For the ADI scheme we use the splitting formula

$$\mathbf{F}(\mathbf{U}) = \mathbf{F}_x(\mathbf{U}) + \mathbf{F}_y(\mathbf{U}),$$

where $\mathbf{F}_x$ and $\mathbf{F}_y$ represent the space discretization terms containing the x- and y-derivatives, respectively. For the Burgers equations such a splitting is possible, because there are no mixed derivatives. So, the ADI scheme for (3.5) is given by [11]

$$\mathbf{U}^{n+1/2} = \mathbf{U}^n + \tfrac{1}{2}\tau[\mathbf{F}_x(\mathbf{U}^{n+1/2}) + \mathbf{F}_y(\mathbf{U}^n)], \tag{3.12a}$$

$$\mathbf{U}^{n+1} = \mathbf{U}^{n+1/2} + \tfrac{1}{2}\tau[\mathbf{F}_x(\mathbf{U}^{n+1/2}) + \mathbf{F}_y(\mathbf{U}^{n+1})]. \tag{3.12b}$$

Note that (3.12a) is explicit in the y-direction and implicit in the x-direction, and vice versa in (3.12b). Since there is a 3-point coupling in each direction, the ADI scheme can be implemented such that only nonlinear tridiagonal systems must be solved at each step.

In order to obtain linear systems, the terms $\mathbf{F}_x(\mathbf{U}^{n+1/2})$ in (3.12a) and $\mathbf{F}_y(\mathbf{U}^{n+1})$ in (3.12b), which can be written in the form (cf. (3.3))

$$\text{(3.13a)} \qquad \mathbf{F}_x(\mathbf{U}^{n+1/2}) = A(U^{n+1/2})\mathbf{U}^{n+1/2} \quad \text{and} \quad \mathbf{F}_y(\mathbf{U}^{n+1}) = B(V^{n+1})\mathbf{U}^{n+1},$$

are linearized as follows:

$$\text{(3.13b)} \qquad \mathbf{F}'_x(\mathbf{U}^{n+1/2}) = A(U^*)\mathbf{U}^{n+1/2} \quad \text{and} \quad \mathbf{F}'_y(\mathbf{U}^{n+1}) = B(V^*)\mathbf{U}^{n+1},$$

with A and B tridiagonal matrices and U^* and V^* approximations to $U^{n+1/2}$ and V^{n+1}, respectively. To maintain second-order accuracy, the approximations U^* and V^* are given by (see [12])

$$U^* = \tfrac{3}{2}U^n - \tfrac{1}{2}U^{n-1}, \qquad V^* = 2V^{n+1/2} - V^n.$$

Now, the ADI scheme only requires the solution of linear tridiagonal systems. In § 4.2 we will discuss some algorithms for the solution of these systems. Due to the linearization process, it is not possible to formulate a fast form for the ADI scheme, as for the OEH scheme (cf. (3.11a) and (3.11c)).

3.3. Stability. Finally, we make some remarks about the stability of both the OEH scheme and the ADI scheme. Consider to this purpose the linear convection-diffusion equation

$$\text{(3.14)} \qquad u_t = -q_1 u_x - q_2 u_y + (u_{xx} + u_{yy})/\mathrm{Re}.$$

Here, the vector $(q_1, q_2)^T$ represents a constant velocity. Now suppose that for the space discretization we use standard central differences, with constant grid sizes h and k in x- and y-direction, respectively. Then von Neumann stability analysis applied to the OEH scheme (3.10) yields the following necessary and sufficient timestep restriction [14], [15] for (3.14):

$$\tau^2\left(\frac{1}{h^2} + \frac{1}{k^2}\right)(q_1^2 + q_2^2) \leqq 4.$$

This inequality shows that the OEH scheme is conditionally stable ($\tau = O(h)$), independent of Re. The ADI scheme for the linear equation (3.14) is unconditionally stable in the sense of von Neumann stability [10].

Remark. A drawback of the OEH scheme is the so-called Du Fort-Frankel (DFF) deficiency [14], [15]. By this we mean that for τ, h, $k \to 0$, the solution of the OEH scheme converges to the solution of the problem

$$\text{(3.15)} \qquad \begin{aligned} u_t &= -uu_x - vu_y + (u_{xx} + u_{yy})/\mathrm{Re} - \frac{1}{\mathrm{Re}}\tau^2\left(\frac{1}{h^2} + \frac{1}{k^2}\right)u_{tt}, \\ v_t &= -uv_x - vv_y + (v_{xx} + v_{yy})/\mathrm{Re} - \frac{1}{\mathrm{Re}}\tau^2\left(\frac{1}{h^2} + \frac{1}{k^2}\right)v_{tt}. \end{aligned}$$

In general, for convergence it is thus necessary that $\tau = o(\max(h, k))$.

4. Implementations. In this section we describe implementation techniques for vectorizing both the OEH scheme and the ADI scheme for use on vector computers. It is our goal to implement the schemes in such a way that they perform efficiently on vector computers. We utilize the vector processing concepts discussed in § 2. The programs have been written in the ANSI FORTRAN77. Thus, the resulting software is portable.

4.1. The OEH scheme. The OEH scheme is based upon the alternating use of the forward and backward Euler rule. Because of the 5-point coupling that exists between the variables, the OEH scheme is diagonally implicit (see § 3.2.1). Specifically, the scheme only requires scalar divisions and no nonlinear equations must be solved.

The obvious choice for the ordering of the grid points is the red-black or chess-board ordering, where all the four neighbours of each point belong to another colour. The grid points may be subdivided accordingly into two vectors that contain the red and black points, respectively. The grid points are numbered along horizontal grid lines. The OEH scheme is performed in four stages (see (3.11a)-(3.11d)). For example, in the first stage the values in the red points are updated using the value in the red point itself and old values in neighbouring black points (i.e., the forward Euler rule), then in the second stage the values in the black points are updated using the old value in the black point and new values in red points (i.e., the backward Euler rule). Throughout the code the elements of the two vectors are stored in consecutive memory elements (i.e., stride 1), which is, in general, an advantage on vector computers. Moreover, no data reorderings must be performed.

Note that the two vectors are not confined to one horizontal grid line, but they extend over the whole grid. This was done in order to achieve improved performance through utilization of longer vectors. As a penalty for using those longer vectors, the values in the boundary points are overwritten, thus destroying the correct boundary values. To restore the correct boundary values, these values are stored separately. Moreover, the first and the last grid points of each horizontal line must be of the same colour to maintain the red-black ordering. Thus, the number of grid points in horizontal direction ($=N$) has to be odd.

The OEH scheme requires minimal storage. In our implementation we used only one extra array of length $NM/2$, which is one fourth of the total number of unknowns. Hence, the total storage amounts approximately to $2.5NM$ memory locations.

4.2. The ADI scheme. The ADI scheme for two-dimensional problems requires the solution of tridiagonal systems along horizontal and vertical grid lines, respectively. Tridiagonal systems form an important class of linear algebraic equations. Consequently, efficient algorithms have been developed for the solution of such systems. The tridiagonal systems can be viewed in various ways. For example, for (3.12a) we have M tridiagonal (linear) systems of order N. The first method we use to solve this system is the Gaussian elimination method. Since the M systems are uncoupled, we can vectorize across the systems, thus resulting into vector operations of length M. On the other hand, the M individual systems can be combined in a single tridiagonal system of order NM to obtain longer vectors. As a consequence, extra memory is needed. Due to the large memory capacities of today's vector computers, it is possible to execute programs with large memory requirements. For example, on the Cyber 205 and the Cray X-MP/24 the maximal memory size is about four million 64-bit words.

Several methods have been proposed to achieve efficient methods for such large systems on vector computers. In this report we use a variant of the partition method of Wang [3], [9], which will be discussed briefly now. First, the tridiagonal matrix is partitioned into a $l \times l$ block tridiagonal matrix with each block a $m \times m$ matrix. The method starts by reducing the tridiagonal system to a tridiagonal system of order l using vector operations. Then the reduced system is solved by Gaussian elimination.

Finally, the other unknowns are solved by back substitution using again vector operations. Although the variant of the partition method has a higher operation count than the Gaussian elimination method, this is an efficient method on vector computers since the vector length of the operations is much higher.

For this variant of the partition method, it is plausible that the off-diagonal elements of the reduced system are very small relative to the main diagonal elements [3], [5], which is confirmed by numerical experiments. Following Van der Vorst [16] and De Goede and Wubs [3], the solution of the reduced tridiagonal system is approximated by a truncated Neumann series. The resulting explicit-implicit method is advantageous for use on vector computers. The price to be paid for the approximation of the reduced system is a possible drop in accuracy. However, due to the relatively small off-diagonal elements, this approach hardly affects the accuracy.

As said in § 2, for the performance on vector computers the data structure is very important. For the ADI scheme, tridiagonal systems must be solved along horizontal grid lines and vertical grid lines, respectively. If the arrays are ordered horizontally, then the x-differences can be calculated efficiently. Likewise, if the arrays are ordered vertically, then the y-differences can be calculated efficiently. These two orderings imply that during the performance of the ADI scheme, reorderings must be performed to change from horizontal to vertical lines and vice versa. The reordering operations have been implemented as efficiently as possible.

Moreover, during the solution of the tridiagonal systems, the variant of the partition method requires vector operations with stride m. The Cray-computer is hardly hampered by a stride unequal to one. However, the CDC Cyber 205 requires contiguous vectors (i.e., stride 1). Therefore, compress/expand instructions are necessary to restructure the vectors. The alternative is to reorder in advance the data structure to obtain contiguous vectors. On the CDC Cyber 205 this alternative may be useful. Both versions have been implemented.

For each of the implementations, the storage requirements are significantly larger than for the OEH scheme, viz. about $9NM$ memory locations.

Summarizing, for the Peaceman-Rachford ADI scheme the following implementations for the solution of the tridiagonal systems have been used:

ADIGE	(Gaussian elimination),
ADIW	(a variant of the partition method of Wang (stride m)),
ADEI	(the explicit-implicit scheme (stride m)),
ADIW1	(ADIW with an extra reordering of the data structure (stride 1)),
ADEI1	(ADEI with an extra reordering of the data structure (stride 1)).

5. Performance. In this section we report on the accuracy and performance of the OEH scheme and the ADI scheme on vector computers. For this purpose, we have applied the schemes to a moving wave front problem. In general, moving wave front problems are difficult to compute since the solution contains sharp gradients, both in space and time. This necessitates the use of small timesteps and, when employing a uniform grid, a small grid size. Therefore, such problems are time and memory consuming and the application of powerful computers (such as, e.g., vector computers) is obvious.

In our experiments the following vector computers and FORTRAN compilers have been used:

(i) (2-pipe) CDC Cyber 205 (SARA, Amsterdam, the Netherlands), max. 200 MFLOP/s, FORTRAN 200 compiler, (the VAST (version 1.22W) precompiler of Pacific Sierra Research Corporation is used).

(ii) Cray X-MP/24 (Cray Research, Bracknell, UK), max. 235 MFLOP/s. FORTRAN CFT77 (version 1.3) compiler.

An exact solution of the Burgers equations can be generated by using the Cole-Hopf transformation [2],

$$u=\frac{2}{\mathrm{Re}}\frac{\phi_x}{\phi} \quad \text{and} \quad v=-\frac{2}{\mathrm{Re}}\frac{\phi_y}{\phi}, \tag{5.1a}$$

where ϕ is the solution of

$$\phi_t=\frac{1}{\mathrm{Re}}(\phi_{xx}+\phi_{yy}). \tag{5.1b}$$

In our test problem we choose $\phi=f_1+f_2$ [18], with

$$\begin{aligned} f_1(x,y,t)&=\exp((-12(x+y)+9t)*\mathrm{Re}/32),\\ f_2(x,y,t)&=\exp((-4(x+2y)+5t)*\mathrm{Re}/16), \end{aligned} \tag{5.2}$$

which yields the exact solution

$$u=\frac{1}{4}*\frac{3f_1+2f_2}{f_1+f_2}=\frac{3}{4}-\frac{1}{4}*\frac{1}{1+\exp((-4x+4y-t)*\mathrm{Re}/32)}, \tag{5.3a}$$

$$v=\frac{1}{4}*\frac{3f_1+4f_2}{f_1+f_2}=\frac{3}{4}+\frac{1}{4}*\frac{1}{1+\exp((-4x+4y-t)*\mathrm{Re}/32)}. \tag{5.3b}$$

The solution represents a wave front at $y=x+0.25t$. The speed of propagation is $0.125\sqrt{2}$ and is perpendicular to the wave front. For increasing values of Re, the wave front becomes sharper. In Fig. 2 the exact solution for u is shown at $t=2.5$ for Re = 100; 1,000; 10,000.

With the purpose of testing the (order of) accuracy of the schemes, we first compare the exact solution of the Burgers equations with the numerical solution obtained for grid sizes $h=k=1/17$, $1/33$, $1/65$, $1/129$ and for timesteps $\tau=1/10$, $1/20$, $1/40$, $1/80$, $1/160$, $1/320$ (provided that the time integration is stable). The computational domain is $\Omega=[0,1]\times[0,1]$ and the time integration interval is $[0,2.5]$. We prescribe time-dependent Dirichlet boundary conditions that are taken from the exact solution and we choose Re = 100. For the time integration we use the OEH scheme, the ADIW scheme, and the ADEI scheme (see § 4).

To measure the accuracy of the numerical solution we define

$$\mathrm{cd}_\infty=-{}^{10}\log(\|\text{global error at } t=2.5\|_\infty), \tag{5.4}$$

denoting the number of correct digits in the numerical approximation at the endpoint $t=2.5$.

Since $\max|u(x,y,t)|=0.75$ and $\max|v(x,y,t)|=1.0$, von Neumann stability analysis applied to the OEH scheme suggests the timestep restriction

$$\tau\leqq\tfrac{4}{5}\sqrt{2}h. \tag{5.5}$$

In Table 5.1 we list the cd_∞-values for all three schemes. We only list the cd_∞-values for the u-field; for the v-field we obtain nearly the same results.

First consider the OEH scheme. From Table 5.1 we can conclude the following:

(i) For small timesteps (e.g. $\tau=1/320$) the time integration error is neglectable, and we can observe the second-order behaviour in space (${}^{10}\log(4)\approx0.6$). On a fine grid (e.g., $h=1/129$) we can observe the second-order behaviour in time since the space discretization error is neglectable.

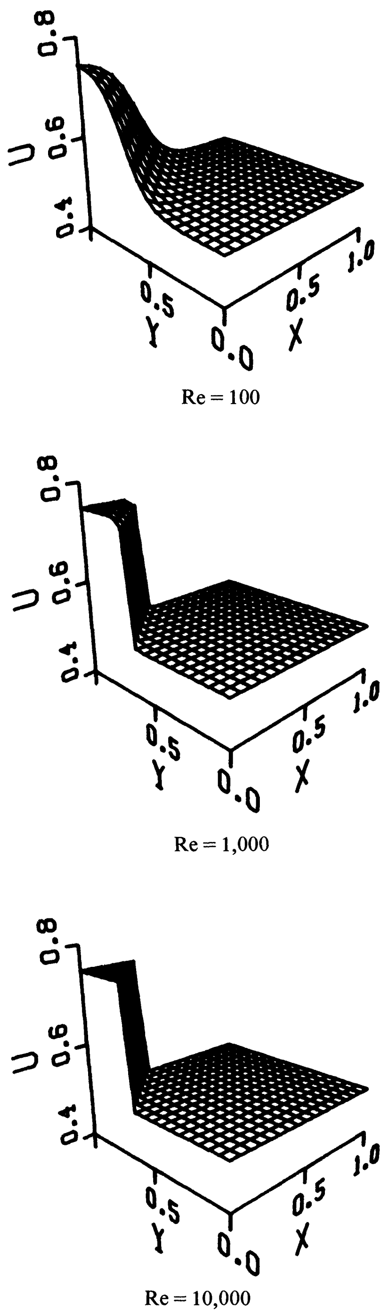


FIG. 2. *Exact solutions* (5.3a) *for* Re = 100; 1,000; 10,000.

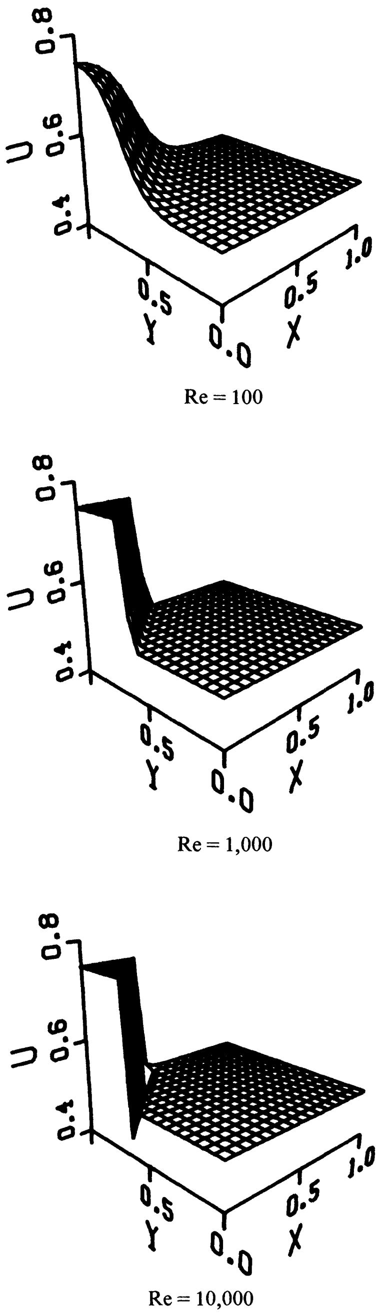


FIG. 3. *Corresponding numerical solutions.*

(ii) For τ fixed and $h \to 0$ the accuracy decreases if τ/h is sufficiently large. This is caused by the DFF deficiency (cf. (3.15)).

(iii) When looking along diagonals (τ/h constant) we observe a second-order behaviour if τ/h is small enough. For larger values of τ/h the scheme fails to converge due to the DFF deficiency.

TABLE 5.1
cd_x*-values for the* OEH, ADIW, *and* ADEI *schemes.*

		Correct digits for u-field (∞-norm)					
Scheme	h^{-1}	$\tau=1/10$	$\tau=1/20$	$\tau=1/40$	$\tau=1/80$	$\tau=1/160$	$\tau=1/320$
OEH	17		2.54	2.55	2.55	2.55	2.55
	33			3.05	3.21	3.20	3.20
	65			2.82	3.37	3.73	3.85
	129				2.84	3.44	3.98
ADIW	17	1.92	2.29	2.48	2.51	2.52	2.52
	33	2.10	2.56	2.89	3.07	3.15	3.18
	65	2.24	2.75	3.19	3.51	3.68	3.77
	129	2.25	2.77	3.26	3.67	3.99	4.19
ADEI	17	1.92	2.29	2.48	2.51	2.52	2.52
	33	2.12	2.57	2.89	3.07	3.15	3.18
	65	2.24	2.75	3.19	3.51	3.68	3.77
	129	2.25	2.78	3.26	3.67	3.98	4.17

Now, consider both ADI-type schemes. In the same way as for the OEH scheme, we can observe second-order behaviour in space and time. In general, the accuracy of the OEH scheme is comparable with that of the ADI-type schemes. However, especially on the finest grid the ADI-type schemes are more accurate than the OEH scheme, because the latter suffers from the DFF deficiency. Note that the accuracy results for the ADIW scheme and the ADEI scheme are comparable. So, the accuracy is hardly reduced if the tridiagonal systems are solved by the approximating method.

Table 5.2 presents the execution times obtained for a single example, namely, for a 129×129 grid with $t=2.5$, $\tau=1/80$, and $\mathrm{Re}=100$. We compare the OEH scheme with the five implementations of ADI-type schemes (see § 4). As an illustration, the implementations have also been performed without vectorization on the CDC Cyber 205 (scalar code). In parentheses we list the ratio in performance of the vectorized code to the scalar code. We emphasize that Table 5.2 contains the execution times for the computation of 200 timesteps without paying attention to the accuracy or stability.

From this experiment we can draw the following conclusions:

(i) On both vector computers the OEH scheme is considerably faster than the implementations of the ADI-type schemes. This is due to the fact that no systems of equations must be solved and no data reorderings must be performed. For the scalar code, it is fair to say that the ratio of the execution time for the ADI-type schemes

TABLE 5.2
Execution times in seconds for a 129×129 *grid with* $t=2.5$, $\tau=1/80$ *and* $\mathrm{Re}=100$.

	Execution times (in seconds)		
Scheme	Cyber 205 (vectorized code)	Cray X-MP/24	Cyber 205 (scalar code)
OEH	1.8	1.0	15.4 (8.6)
ADIGE	15.0	6.0	118.2 (7.9)
ADIW	27.3	8.7	181.6 (6.6)
ADEI	22.4	6.1	176.6 (7.8)
ADIW1	18.6	8.9	187.1 (10.0)
ADEI1	12.6	6.7	182.2 (14.4)

to the OEH scheme is misleading. For example, the data reorderings (from x-ordering to y-ordering and vice versa) are uneconomical for use on scalar computers. So, the scalar code for the ADI-type schemes is far from optimal.

(ii) On the CDC Cyber 205 the vectorized code is much faster than the scalar code. For all schemes a considerable speed-up factor is obtained.

(iii) On the CDC Cyber 205 it is beneficial to reorder the data structure to obtain contiguous vectors (compare ADIW with ADIW1 and ADEI with ADEI1). The speed-up in performance justifies the overhead due to the data reordering. On the Cray X-MP/24 this does not hold since the Cray is hardly hampered by a stride unequal to one.

(iv) In general, the Cray X-MP/24 is considerably faster than the (2-pipe) CDC Cyber 205. This is due to a smaller clock cycle and a better compiler.

(v) On the CDC Cyber 205 the fastest method is ADEI1 (i.e., the explicit-implicit method with an extra reordering of the data structure). On the CRAY X-MP/24 however, the Gaussian elimination method and the explicit-implicit method ADEI are the fastest methods.

Finally, we examine the accuracy behaviour of the OEH scheme and the ADIW scheme for increasing values of Re. In this experiment we compute the numerical solution at $T=2.5$ and use the grid size values $h=k=1/33$, $1/65$, $1/129$, $1/257$. Especially for large values of Re we may expect oscillations in the solution. Therefore, the cd_∞-value, as defined in (5.4), is a too strict measure for the accuracy. Instead, we define

$$\mathrm{cd}_1 = -{}^{10}\log\,(\|\text{global error at } t=2.5\|_1).$$

We start our computations for $\mathrm{Re}=100$ on a 33×33 grid. On each grid and for each Re-number we choose the timestep as large as possible such that $\mathrm{cd}_1 \geqq 3$. As soon as $\mathrm{cd}_1 < 3$ for each timestep we switch to the next finer grid and choose an appropriate timestep. In Table 5.3 we list the cd_1-values for the u-field for increasing values of Re; for the v-field we find nearly the same results. For the ADIW scheme the timestep is listed in parentheses. In this experiment we used the ADIW scheme; however, nearly the same results would have been obtained for the ADEI scheme.

TABLE 5.3
cd_1-*values for the* OEH *and* ADIW *scheme for increasing values of* Re.

	Correct digits for u-field (1-norm)							
	OEH scheme				ADIW scheme			
Re	$h=1/33$ $\tau=1/40$	1/65 1/80	1/129 1/160	1/257 1/320	$h=1/33$	$h=1/65$	$h=1/129$	$h=1/257$
100	4.05				3.30 (1/10)			
500	3.08				2.95 (1/80)	3.37 (1/40)		
1,000	2.51	3.42				3.19 (1/80)		
1,500		3.06				2.91 (1/160)	3.36 (1/80)	
2,000		2.86	3.74				3.15 (1/80)	
3,000			3.39				3.09 (1/160)	
4,000			3.18				2.81 (1/160)	3.01 (1/80)
5,000			3.03					3.23 (1/160)
6,000			2.88	3.70				3.32 (1/320)
7,000				3.59				
10,000				3.35				

From Table 5.3 we can conclude the following:

(i) In order to obtain the prescribed accuracy, the ADI scheme requires in general a finer grid than the OEH scheme. This is possibly due to the linearization process of the ADI scheme (see (3.13)). Both schemes require a comparable timestep. So, for large Re-numbers the OEH scheme seems to be more suitable than the ADI-type schemes for the numerical solution of the Burgers equations, at least for the present type of solution.

(ii) The DFF-deficiency of the OEH scheme is virtually absent for large Re-numbers since the terms u_{tt}/Re and v_{tt}/Re are very small, except in a small region near the wave front (see (3.15)).

In Fig. 3 we present the numerical solution for the u-field for $\mathrm{Re}=100$; 1,000; 10,000 computed with the OEH scheme.

6. Concluding remarks. In this paper we examined the efficiency and performance of the odd-even hopscotch (OEH) scheme and the alternating direction implicit (ADI) scheme on vector computers, viz. the CDC Cyber 205 and the Cray X-MP/24. For the ADI scheme the following methods for the solution of the tridiagonal linear systems have been used: the Gaussian elimination method (ADIGE), a variant of the partition method of Wang (ADIW) and the explicit-implicit method (ADEI). The vectorized codes were considerably faster than the corresponding scalar codes.

First, let us consider the advantages of the OEH scheme over the ADI-type schemes:

(i) On both vector computers the OEH scheme is considerably faster than the ADI-type schemes, due to the near-explicitness of the OEH scheme.

(ii) The OEH scheme has minimal storage requirements. In our implementations we used about four times more memory space for the ADI-type schemes than for the OEH scheme. This is due to the way in which the tridiagonal systems are solved (see § 4).

(iii) It is very easy to implement the OEH scheme for both linear and nonlinear problems. For the ADI-type schemes the nonlinear tridiagonal systems of equations must be linearized in some way (cf. (3.13)). Moreover, the OEH scheme can be extended to multidimensional problems in a straightforward manner, contrary to the ADI-type schemes.

The ADI-type schemes have the following advantages over the OEH scheme:

(i) The ADI-type schemes have a better stability behaviour than the OEH scheme.

(ii) The OEH scheme suffers from the Du Fort-Frankel (DFF) deficiency that in general, has a negative influence on the accuracy.

Comparing the ADI-type schemes, the ADEI scheme and the ADIGE scheme have a comparable performance on vector computers. However, for test problems with an irregular domain, the ADEI scheme is to be preferred since in that case vectorization across the systems requires extra operations. In the near future, we will extend the codes for application to the incompressible Navier-Stokes equations.

Acknowledgments. We wish to express our gratitude to the ZWO Werkgroep Gebruik Supercomputers (WGS) for providing the necessary computer time on the CDC Cyber 205 and the Cray X-MP/24.

REFERENCES

[1] CDC Cyber 200 FORTRAN reference manual, version 1, publ. number 60480200H, Sunnyvale, CA.

[2] C. A. J. Fletcher, *A comparison of finite element and finite difference solutions of the one- and two-dimensional Burgers equation*, J. Comput. Phys., 51 (1983), pp. 159–188.

[3] E. D. DE GOEDE AND F. W. WUBS, *Explicit-implicit methods for time-dependent partial differential equations*, Report NM-R8703, Centre for Mathematics and Computer Science, Amsterdam, the Netherlands, 1987.

[4] A. R. GOURLAY, *Hopscotch: A fast second-order partial differential solver*, J. Inst. Maths. Appl., 6 (1970), pp. 375-390.

[5] D. HELLER, *Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems*, SIAM J. Numer. Anal., 13 (1976), pp. 484-496.

[6] R. W. HOCKNEY AND C. R. JESSHOPE, *Parallel Computers: Architecture, Programming and Algorithms*, Adam Hilger, Bristol, 1981.

[7] P. J. VAN DER HOUWEN AND J. G. VERWER, *One-step splitting methods for semi-discrete parabolic equations*, Computing, 22 (1979), pp. 291-309.

[8] J. VAN KAN, *A second-order pressure correction method for viscous incompressible flow*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 870-891.

[9] P. H. MICHIELSE AND H. A. VAN DER VORST, *Data transport in Wang's partition method*, Parallel Computing, 7 (1988), pp. 87-95.

[10] A. R. MITCHELL AND D. F. GRIFFITHS, *The Finite Difference Method in Partial Differential Equations*, John Wiley, Chichester, 1980.

[11] D. W. PEACEMAN AND H. H. RACHFORD, JR., *The numerical solution of parabolic and elliptic differential equations*, J. Soc. Indust. Appl. Math., 3 (1955), pp. 28-41.

[12] B. P. SOMMEIJER, *An* ALGOL 68 *implementation of two splitting methods for semi-discretized parabolic differential equations*, Report NM-NN 15/77, Centre for Mathematics and Computer Science, Amsterdam, the Netherlands, 1977.

[13] R. PEYRET AND T. D. TAYLOR, *Computational Methods for Fluid Flow*, Springer-Verlag, Berlin, New York, 1983.

[14] J. H. M. TEN THIJE BOONKKAMP, *The odd-even hopscotch pressure correction scheme for the incompressible Navier-Stokes equations*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 252-270.

[15] J. H. M. TEN THIJE BOONKKAMP AND J. G. VERWER, *On the odd-even hopscotch scheme for the numerical solution of time-dependent partial differential equations*, Appl. Numer. Math., 3 (1987), pp. 183-193.

[16] H. A. VAN DER VORST, *A vectorizable variant of some* ICCG *methods*, SIAM J. Sci. Statist. Comput., 3 (1982), pp. 86-92.

[17] H. H. WANG, *A parallel method for tridiagonal system equations*, ACM Trans. Math. Software, (1981), pp. 170-183.

[18] G. B. WHITHAM, *Linear and Nonlinear Waves*, John Wiley, New York, 1974.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 2, pp. 368-388, March 1990 010

TIMING ANALYSIS OF THE MONOTONIC LOGICAL GRID FOR MANY-BODY DYNAMICS*

J. M. PICONE†, S. G. LAMBRAKOS†, AND J. P. BORIS†

Abstract. The Monotonic Logical Grid (MLG) data structure is compared to alternative data structures for tasks relevant to computing the dynamics of N moving objects. The tasks include identifying near neighbors of designated nodes, retrieving information on the near neighbors, and ordering this information according to distances of the associated near neighbors from the designated nodes. The test model consists of a collection of $N = 64\text{K}$ (65,536) noninteracting objects with randomly initialized velocities. The calculations took place on the Naval Research Laboratory (NRL) Cray X-MP computer, which has a hardware gather-scatter capability. The comparisons include two types of alternative data structures for which the indexing is static (the data corresponding to each node always have the same index and memory locations). Data structure "Type 1" carries no additional information or "pointers" that could identify the near neighbors of each node. Data structure "Type 2" *does* carry coarse information on near neighbors by maintaining linked lists or a related system of "pointers" that change with the motion of nodes in time. The numerical tests presented show that the MLG data structure is vastly superior to the Type 1 data structure when near-neighbor information is needed for a large number N_f of designated or "focal" nodes. This occurs because the process of identifying near neighbors requires N_f identical sorting or partitioning processes per frame (or timestep) in the case of Type 1 data structures while only one sort per frame is necessary to maintain the ordering of data in the MLG data structure. In addition, the MLG is superior for the task of finding some number M_{nn} of the *nearest* neighbors for each of N_f focal nodes, where $M_{nn} \ll N$ and $N_f \sim N$. The MLG provides several advantages over Type 2 data structures, even though the respective operation counts are quite similar. The advantages include efficiency of memory allocation and memory management, smaller memory requirements, vectorizability and parallel partitioning on a wide variety of computer architectures, and simplicity of programming. The last property should lead to computer software with a reduced error rate and code that is more amenable to revision.

Key words. monotonic logical grid, dynamic data structure, near-neighbors problem

AMS(MOS) subject classifications. 65K05, 68A10, 68A20

1. Introduction.

The Monotonic Logical Grid (MLG) algorithm organizes data associated with a collection of nodes so that spatial or geotemporal near neighbors will also be near neighbors in index space (Boris [2]). Appendix I gives a simple example showing how an MLG might be constructed given a set of nodes with arbitrary spatial locations. Arranging data according to an MLG vastly reduces the work required for any calculation in which the near neighbors of each node must be identified or analyzed. We find examples of such requirements in molecular dynamics (Hockney and Eastwood [3], Lambrakos and Boris [5]) and sensor data analysis and target tracking and correlation (e.g., Reid [8]). The restructuring portion of the MLG algorithm, which maintains the proper ordering of memory, is of computational complexity $N \log N$, where N denotes the number of nodes. In addition, this iterated restructuring procedure is ideally suited for highly parallel and vector computers.

The present study compares the MLG data structure directly with two other types of data structures that might be used for the same tasks. For each of these alternative data structures, the data associated with each node (e.g., spatial and velocity coordinates) remain in the same memory locations throughout the evolution of the system. The term "static" will represent this type of memory allocation. The first type of data

* Received by the editors October 13, 1986; accepted for publication (in revised form) August 3, 1987. This research was supported by the Office of Naval Research and the Strategic Defense Initiative Organization.

† Laboratory for Computational Physics, U.S. Naval Research Laboratory, Washington, D.C. 20375.

structure maintains no additional information or "pointers" identifying the near neighbors of each node. A model with a "Type 1" data structure might determine near neighbors of a given node by computing the $N-1$ distances to the other nodes and then ordering these distances according to a particular definition of *near* neighbors. Another method would be to select those nodes whose spatial coordinates fall within a prescribed interval about the coordinates of each designated or "focal" node. To illuminate the differences between this "Type 1" approach and the MLG, when implemented on a modern, high-speed computer, we have run a series of tests on the NRL Cray X-MP with a system containing 64K (65,536) nodes. The initial conditions involve regular spacing of the nodes on a cubical lattice with random initial velocities. This favors the MLG less than would a situation in which the nodes were all moving in roughly the same direction at roughly the same speeds. The present discussion covers the following tests on the MLG and a representative Type 1 data structure:

(1) Find the indices of a specified number of near neighbors of given designated or "focal" nodes and write these to a buffer. The Type 1 data structure implements "Test 1" by computing the $N-1$ distances of the nodes from each focal node and then ordering these distances from the smallest to the largest. To bracket the results, we include two other, less comprehensive, and less expensive definitions of near neighbors for the Type 1 data structure. "Test 1A" identifies those neighboring nodes that are within a prescribed distance of each focal node. "Test 1B" identifies those nodes whose spatial coordinates fall within a prescribed interval about the spatial coordinates of each focal node. The above three definitions of "near neighbors" for the Type 1 data structure carry varying amounts of implicit information on the vicinities of the focal nodes.

(2) Write the data associated with the near neighbors identified in Test 1 to a buffer. This requires random sifting through the Type 1 (and Type 2) data structures and involves inefficient short loops with the MLG, when implemented on the Cray X-MP.

(3) Order the data on the sets of near neighbors identified in Test 1 according to distance of each near neighbor from a given focal node. Test 3 thus determines the *nearest* neighbors of each focal node.

The next section presents the results of these tests and converts the data to equations for predicting the relative performance of the two approaches.

The second class of data structures that are considered as alternatives to the MLG are also static. However, these "Type 2" data structures maintain some dynamic information or "pointers" for identifying the near neighbors of each node. An excellent example is the method of Hockney and Eastwood [3], in which the physical space is divided into cells and linked lists are used to identify the particular nodes located in each cell at a particular time. As a given node passes from one cell to a neighboring cell, the index of the node disappears from the list associated with the former cell and appears in the list corresponding to the cell that the node has just entered. Prior to the MLG, this was one of the best methods of computing near-neighbor interactions. The reader who is familiar with the MLG concept will recognize this alternative as being a distant relation of the MLG algorithm. Rather than moving the node data in memory to maintain the knowledge of spatial or geotemporal relationships among nodes, Hockney's Type 2 data structure essentially swaps information (node indices) between linked lists. The linked lists associate data that have a fixed location in memory to fixed cells in space. These linked lists represent extra memory and a scalar component of the restructuring computation that is absent with the MLG. The price paid for the benefits of the MLG data structure is the need to move much more data than the linked lists require. In addition to Type 2 data structures used primarily for many-body dynamics, some of the more general near-neighbor search algorithms are relevant to our discussion. The k nearest-neighbor finding algorithm developed and analyzed statistically by Kim and Park [4], in fact, bears some resemblances to the MLG. Section 3 explores Type 2 data structures. Section 4 provides our conclusions concerning the data structures that have been considered.

2. Monotonic logical grid vs. "Type 1" data structures.

2.1. Description of tests. The test problem run on the Cray X-MP consists of 64K noninteracting nodes moving randomly in a cubical volume. The boundary conditions in each x-y plane are skew-periodic (Lambrakos and Boris [5]), and the model uses reflecting boundary conditions perpendicular to the x-y planes at each end of the z-domain. We prevent boundary conditions from influencing our results by choosing focal nodes that lie near the center of an x-y plane and that have a sufficient displacement from the lowest and highest z-values in the computational domain. Each node has a set of MLG indices and a single, constant Type 1 index or identification (ID) number. As the system evolves, the MLG indices of a node will change while the ID will not. We may thus interpret ID as a label of the static memory locations of the data on the nodes.

For the Type 1 data structure, the calculation of distances between a focal node at (x_f, y_f, z_f) and all others is fully vectorized. For Test 1, the code actually sorts on the square of the internode distance

$$R_{\rm id}^2 = (x_{\rm id} - x_f)^2 + (y_{\rm id} - y_f)^2 + (z_{\rm id} - z_f)^2 \tag{1}$$

rather than the distance itself. This saves time by eliminating the additional square root operation required for computing the distance. In (1), the subscript f denotes the focal node, and the subscript id is the index of running over the other nodes in this system.

For Test 1 of the Type 1 data structure, the HEAPSORT algorithm (Nijenhuis and Wilf [7]) orders nodes according to $R_{\rm id}^2$. HEAPSORT has order $N \log N$ computational complexity and is a standard algorithm in the theory of sorting. Our routine is not vectorized. For Test 1 the ordering of nodes by HEAPSORT accounts for most of the time expenditure, so that a significantly faster sorting algorithm would yield correspondingly reduced time costs for Test 1. Fully vectorizing on the Cray X-MP typically yields an improvement in speed by a factor of 5 to 10. The restructuring algorithm required to maintain the MLG is also not vectorized in the present tests. The relative timings should therefore be meaningful even if the absolute computation rates might not. Note that, for the Type 1 data structure, Test 3 above ("find the *nearest* neighbors") is automatically satisfied on completion of Test 1.

Test 1A involves a comparison of the quantities $R_{\rm id}^2$ to R^2, the square of a specified radius vector centered at each focal node. The values used here are $R = 2\delta$, 3δ, 4δ, and 5δ, where δ is a constant that is approximately equal to the average difference in the x, y, or z coordinates of adjacent nodes. Test 1B searches through the coordinates $(x_{\rm id}, y_{\rm id}, z_{\rm id})$ to find the nodes whose locations fall within the interval from $(x_f - m\delta, y_f - m\delta, z_f - m\delta)$ to $(x_f + m\delta, y_f + m\delta, z_f + m\delta)$, where m is an integer. The runs of Test 1B used $m = 2$ and $m = 4$.

In the case of Type 1 data structures, the retrieval of data associated with the near neighbors (Test 2) after the sorting is completed requires random sifting through the data structure. For a particular focal node and timestep, the IDs of the near neighbors will be random because of the random motion of the particles. This random sifting is called a "gather" operation, for which the NRL Cray X-MP has a special hardware capability. Thus, the Type 1 data structure can attain an appreciable fraction of vector speed when retrieving data associated with the near neighbors. This apparent gain relative to the MLG will be difficult to sustain in a fully parallel computing environment.

The time expenditure in using the MLG data structure depends on the following:

(1) How much swapping must be done at each timestep or "frame" to maintain the indexing in MLG order.

(2) The number of "near neighbors" that must be accessed and processed for each focal node. Because of the MLG organization of computer memory (index space), the near neighbors and the focal node will have a contiguous set of indices.

(3) The ability of a given computer system to capitalize on the fact that a contiguous set of indices identifies the data corresponding to near neighbors of a given node. Even on a conventional scalar machine this provides an advantage in that the near-neighbor data may be accessed through the use of DO-loops without performing a "gather" operation.

To be more specific, denote the indices of a focal node by the set of integers (i_f, j_f, k_f). To define the set of the near neighbors, we must specify the size of the index interval from $(i_f - \Delta i,\ j_f - \Delta j,\ k_f - \Delta k)$ to $(i_f + \Delta i,\ j_f + \Delta j,\ k_f + \Delta k)$, within which the indices of "near neighbors" will fall. That is, we require only the set of integers $(\Delta i, \Delta j, \Delta k)$ that define the "maximum index offsets" of the set of near neighbors.

The appropriate maximum index offsets to use in retrieving data depend primarily on the number of *nearest* neighbors that the user wishes to identify. As shown in previous papers, the MLG provides coarse quantitative information on the identities of the nodes that are closest spatially to a focal node. This information is coarse because displacement in index space does not correspond perfectly to geometric or geotemporal displacement. Some of the nodes within a given index interval (as defined above) might actually be farther from the focal node than some of the nodes whose indices fall outside the interval. Thus we must be careful to retrieve data from a sufficiently large interval to include the desired number of *nearest* neighbors. Fortunately the cost of this safety factor is minor in practice, since the number of near neighbors that must be considered will only be a small fraction of the total number of nodes N, when N is large (Lambrakos and Boris [5]).

Given the dependence of the MLG data structure performance on (1) and (2) above, two parameters provide the basis for delineating our tests. The first is a dimensionless time interval between frames, given by

$$\text{DT} = \frac{v_r \delta t}{\delta s}. \tag{2}$$

Here v_r is a characteristic relative velocity component of two neighboring nodes. For the present tests, v_r is the maximum value of a given velocity component (e.g., x-component) that any node could have. In (2), δt is the value of the time interval between frames in the simulation, and δs is the minimum average distance between adjacent particles along each of the three coordinate axes taken separately. Expressed mathematically,

$$\delta s = \min\{\langle \delta x \rangle, \langle \delta y \rangle, \langle \delta z \rangle\}$$

where

$$\langle \delta x \rangle \equiv \frac{1}{(N_x - 1) N_y N_z} \sum_{k=1}^{N_z} \sum_{j=1}^{N_y} \sum_{i=1}^{N_x - 1} |x_{i+1jk} - x_{ijk}|$$

with similar expressions for $\langle\delta y\rangle$ and $\langle\delta z\rangle$. Thus, if the average distance between adjacent nodes along the x-axis is smaller than the similar quantities along the y- and z-axes, the choice is $\delta s = \langle\delta x\rangle$. Note that the above expression for $\langle\delta x\rangle$ is appropriate for randomly moving particles because i, j, and k are MLG indices (see Appendix I).

These definitions of v_r and δs permit us to determine a characteristic time interval over which internode crossings occur. Other definitions are possible and would merely result in a rescaling of DT. The quantity DT indicates the frequency with which nodes will pass each other during a timestep or frame interval. This, in turn, measures how much work (sorting) is required to maintain (or restructure) the MLG after each framing interval. In molecular dynamics calculations, DT is usually less than one (Lambrakos and Boris [5]). The tests have covered values of DT up to 3.0. In our tests, the maximum of any velocity component (e.g., x-component) was 4.0×10^5 cm/s, and the average displacement of nodes along any axis was 1.0×10^{-7} cm. Given this discussion, however, any values of v_r, δt, and δs giving the same values of DT would yield the same conclusions as those described below.

The second group of parameters that affect the results for the MLG data structure is the set of maximum MLG index offsets $(\Delta i, \Delta j, \Delta k)$ defining the vicinity of a given focal node from which the near neighbor data are retrieved. That is, for a given set of MLG focal node indices (i_f, j_f, k_f), the code will retrieve data for near neighbors whose three indices lie in the intervals

$$\begin{aligned} i_f - \Delta i &\leqq i \leqq i_f + \Delta i, \\ j_f - \Delta j &\leqq j \leqq j_f + \Delta j, \\ k_f - \Delta k &\leqq k \leqq k_f + \Delta k. \end{aligned} \tag{3}$$

Using larger offsets requires the computation of more distances and the use of more sorting time to identify the *nearest* neighbors (Test 3). The present tests use offsets of 1, 2, 3, and 5.

In running the tests, we actually used five focal nodes to get an average time expenditure per focal node for each test. The variations in results for the focal nodes were negligible, so that averages based on five nodes are quite adequate. Each calculation ran for 500 frames (timesteps) and performed the tests at intervals of 100 frames to ensure that the ensemble had undergone significant changes due to the random velocities of the nodes. The timings for the various tests were expressed in seconds per frame per focal node.

The MLG restructuring time per frame is independent of the number of focal nodes used, so that the time required *per focal node* is inversely proportional to the number of focal nodes processed. In molecular dynamics calculations, an appreciable number of the nodes in the system must be processed as "focal nodes" at each timestep or frame. Thus, the MLG restructuring time per frame per focal node will be quite small for realistic cases. The test calculations did not include a vectorized MLG restructuring algorithm, so that the MLG results represent upper bounds on the required sorting time. To perform Test 3 for the MLG, the model had to compute distances of the near neighbors from the focal nodes and then render them in ascending order. The code did this with the same HEAPSORT algorithm as that used in ordering the data in the Type 1 structure. Obviously a vectorized sorting module would have done a faster job on this sort as well. In fact, the structure of the MLG itself contains enough information on nearest neighbors so that no sorting should be required to compute the appropriate interactions. Our MLG timing results therefore represent worst cases, again a conservative comparison.

2.2 Test results. Test results are in the form of graphs and tables. Some procedural points are important to understand the discussions. First, the cost of Test 1 (identifying *near* neighbors) should include sorting the nodes to restructure the MLG in order to correspond to Tests 1, 1A, and 1B of the Type 1 data structure. The latter performs Test 1 (Test 1A) by sorting (partitioning) the nodes according to values of R_{id}^2 and performs Test 1B by partitioning nodes according to the values of the spatial coordinates. Our graphs of Test 1 costs includes the respective MLG and Type 1 sorting or partitioning times. However, in Tables 1 and 2 of MLG results, the sorting time is in a separate column because the cost of maintaining the MLG is independent of the number of focal nodes in the problem. The remainder of the cost associated with MLG Test 1 is proportional to the number of focal nodes, as is the time required in the case of the Type 1 data structure for sorting or partitioning according to the Test 1, 1A, or 1B criteria.

The second point concerns the procedure for finding *nearest* neighbors (Test 3). For Test 1 of the Type 1 data structure, the model sorts according to distance from a given focal node and incurs no additional cost for determining *nearest* neighbors. The procedure for finding nearest neighbors in the case of the MLG involves two steps:

(1) Identify enough *near* neighbors to contain the desired number of *nearest* neighbors. The definition of "enough" will determine the maximum index offsets (Δi, Δj, Δk) of near neighbors from each focal node. For the MLG, the maximum offset along each axis should be no more than five for most practical problems.

(2) Order the set of near neighbors according to internode distance R_{id}^2 in (1) to determine the required number of nearest neighbors.

This would also be the procedure for the Type 1 data structure, if the method of Test 1A or 1B were used to find near neighbors. Step (2) is much less expensive than the HEAPSORT performed in Test 1 of the Type 1 structure when the total number of nodes N is large. This is true because, in practice, the number of near neighbors identified in Step (1) is much smaller than N. A final point is that this determination of spatial *nearest* neighbors actually would *not* be performed when the MLG is in use. Rather, a direct parallel calculation of interactions involving the *near* neighbors in index space would most likely be performed. Once again, this is because the *nearest* neighbors form a subset of those nodes already selected through index offsets as *near* neighbors.

2.2.1. Graphs of costs vs. number of focal nodes. Figures 1–3 present the results of our tests on the MLG data structure, and Figs. 4–6 present our results on the Type 1 data structure. The cost of each test is in units of seconds per frame, and the graphs use a log-log scale in all cases. This is because our results cover a wide range in number of designated or "focal" nodes, for which near neighbors are found. In most applications of interest, a large fraction (if not all) of the nodes will be processed as focal nodes. *The reader should exercise caution in comparing the graphs, as the maximum order of magnitude on the vertical scale will vary from figure to figure.*

Figure 1 shows the results of Test 1 (find a specified number of near neighbors and write the MLG indices and ID to a buffer) for the MLG data structure. As indicated above, the near neighbors are selected in groups defined by maximum index offsets from the indices of each focal node (eq. (3)). If the maximum offsets for i, j, and k have the same value Δ, then the number of near neighbors is

$$N_{\text{nn}} = (2\Delta + 1)^3 - 1. \tag{4}$$

As shown in Fig. 1, the calculations used maximum offsets of 1, 3, and 5. In addition, the dimensionless timestep is DT = 0.6, which is reasonable for most problems. At

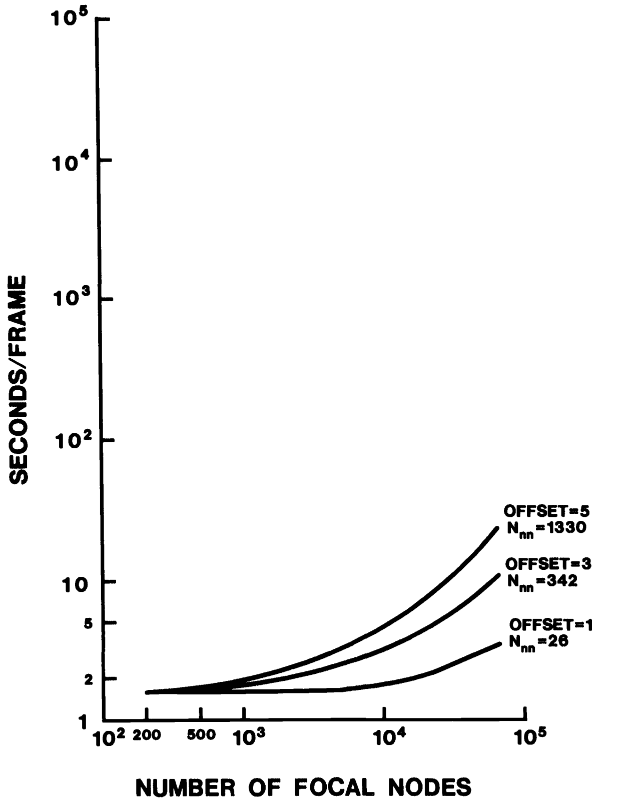


FIG. 1. *Cost per frame of using a Monotonic Logical Grid* (MLG) *data structure on the* NRL *Cray* X-MP *computer. Test* 1: *Identify near neighbors of each focal node. The total number of nodes is* 64K *and the dimensionless time interval between frames is* DT = 0.6.

small numbers of focal nodes, the cost goes asymptotically to the time required to maintain the MLG from frame to frame. This restructuring process depends on DT and N, but not on N_f, the number of focal nodes. We have included this "sorting" time with Test 1 as a natural part of the cost in determining near neighbors. The corresponding sorting time when using the Type 1 data structure constitutes most of the cost of Test 1 in that case. At higher values of N_f, the cost of the MLG near-neighbor access and write begins to increase linearly with N_f.

Figure 2 shows that accessing (retrieving) the data associated with the near neighbors (MLG Test 2) varies linearly with N_f and becomes more expensive than the simple writing of near neighbors indices. This is primarily because the code included a significant number of data arrays: three position coordinates, three velocity coordinates, and two extra words per particle for other information. Note that the vertical scale of Fig. 2 is three orders of magnitude lower than the vertical scale of Fig. 1.

Figure 3 shows the cost of ordering the near neighbors according to distance from their respective focal nodes (MLG Test 3). This would be the final step in determining nearest neighbors using an MLG data structure. However, this is not really necessary with a highly parallel processor, since we could simply perform any "nearest-neighbor" interaction calculation by looping over the near neighbors identified in Test 1. Note that at large N_f, MLG Test 3 costs more than Tests 1 and 2 and that the cost curves depend primarily on the choice of maximum index offset Δ rather than the total number of nodes N. In addition, the cost scales as $N_{nn} \log N_{nn}$ for a given value of N_f.

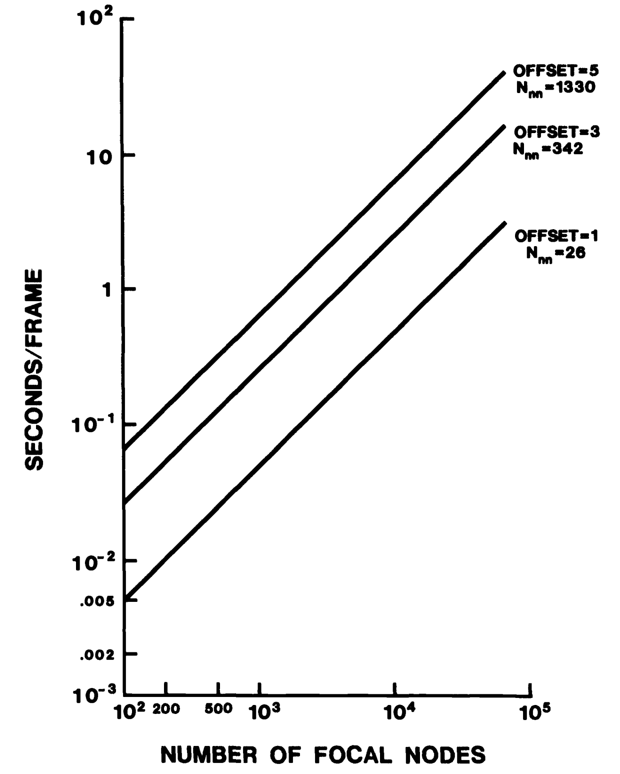


FIG. 2. *Cost per frame of using a Monotonic Logical Grid* (MLG) *data structure on the* NRL *Cray* X-MP *computer. Test* 2: *Access the data associated with the near neighbors of each focal node and write to a buffer. The total number of nodes is* 64K *and the dimensionless time interval between frames is* DT = 0.6.

Figure 4 shows the cost of Test 1 for the Type 1 data structure. To find near neighbors, the model computed the distances of all nodes from a given focal node and then used a HEAPSORT routine to render the distances in ascending order. Thus the code actually computed nearest neighbors (Test 3) at the same time. Note that this process depends on the total number of nodes N along with N_f; the scaling is $N \log N$ because of the use of a HEAPSORT routine. This scaling has permitted us to plot the two dashed curves for smaller values of N. The curves terminate at $N_f = N$. A comparison of Figs. 1 and 4 shows that the cost of this near neighbors calculation for $N = 64$K is more than three orders of magnitude more expensive than accessing near neighbors with an MLG.

Figures 3 and 4 show a difference of 2 to 3 orders of magnitude between the Type 1 data structure (Test 1) and the MLG in the cost of finding *nearest* neighbors when $N = 64$K. This need not be true for significantly smaller values of N, if relatively large index offsets are needed in the MLG for an accurate calculation. Such would be the case if the spatial density of nodes were high enough that a significant fraction of the nodes would influence any focal node. The cost of finding nearest neighbors for $N = 2{,}000$ using the Type 1 data structure is roughly equal to the MLG cost when $\Delta = 5$, since $N_{\mathrm{nn}} = 1{,}330 \sim N$. Experience in molecular dynamics simulations of dimer formation suggests that $\Delta = 4$ is adequate for $N \sim 10^3$ diatomic molecules (Lambrakos et al. [6]). For simulations of similar numbers of inert atoms, $\Delta = 3$ is adequate.

When comparing Figs. 1 and 4, the reader must remember that the MLG timings are based on $N = 64$K nodes. The value of N determines the MLG restructuring time,

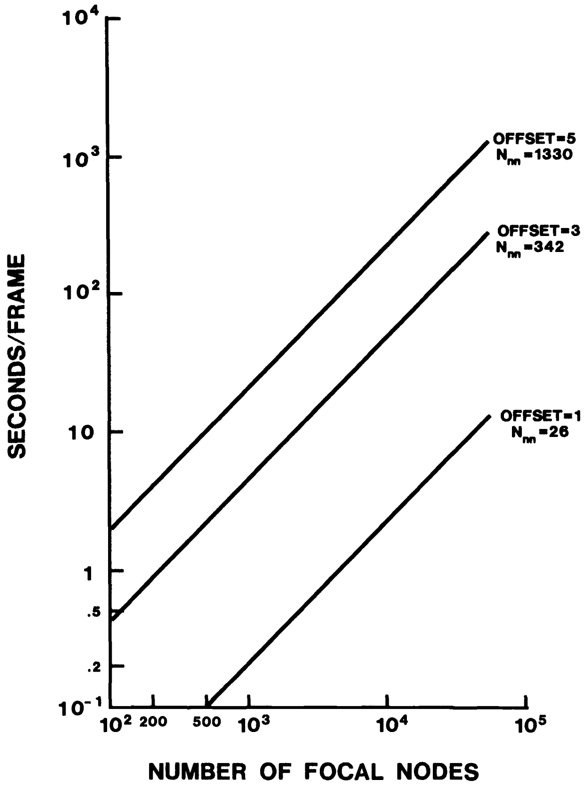


FIG. 3. *Cost per frame of using a Monotonic Logical Grid* (MLG) *data structure on the* NRL *Cray* X-MP *computer. Test* 3: *Order the near neighbors according to distances from each focal node. The total number of nodes is* 64K *and the dimensionless time interval between frames is* DT = 0.6

which is the minimum value indicated on the graph (≈1.6 sec). This restructuring time scales as $N \log N$ and would be much smaller for $N = 2{,}000$, causing the curves to be translated downward on Fig. 1. Thus, going to lower N (dashed curves in Fig. 4) will not give the Type 1 data structure an edge over the MLG.

Figure 5 shows the cost of the alternative methods of identifying near neighbors using the Type 1 data structure. In the method of Test 1A, the partitioning according to distance from each focal node is much simpler than in HEAPSORT. Here we merely ask which nodes are within a prescribed distance of each focal node. This test does *not* provide *nearest* neighbors, in contrast to Test 1 above for the Type 1 data structure. *The information content is also much lower than that in the* MLG *data structure.* Separate runs used four prescribed distances, $R = 2\delta, 3\delta, 4\delta$, and 5δ, as described in § 2.1. The result scales linearly with the total number of nodes N and the number of focal nodes N_f and is almost identical for all values of R. For Test 1B, the code searched for those nodes whose spatial coordinates fell within a prescribed coordinate interval containing a given focal node. We ran the test twice, defining the interval by adding $\pm m\delta$ to each coordinate of the focal node, where m was an integer equal to two and four, respectively. The results were almost identical to those for Test 1A. Equation (5) expresses the cost for Tests 1A and 1B as

$$C_{1A,1B} = N_f \times 0.036 \times (N/65536) \text{ sec/frame.} \tag{5}$$

A comparison of Figs. 4 and 5 for the Type 1 data structure shows that the HEAPSORT

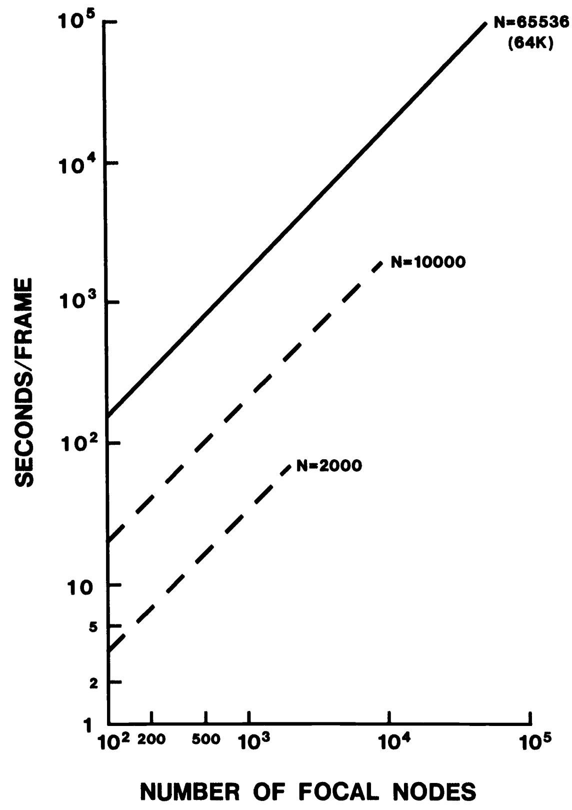


FIG. 4. *Cost per frame of using a conventional* (*Type* 1) *data structure on the* NRL *Cray* X-MP *computer. Test* 1: *Identify near neighbors of each focal node by using a* HEAPSORT *routine to order the distances of all nodes from the focal node. The total number of nodes is* 64K. *The dashed lines give predicted results for* $N = 10{,}000$ *and* 2,000.

(Test 1) costs are over an order of magnitude more than testing to see which nodes are within a prescribed distance of each focal node or a prescribed coordinate interval (Tests 1A and 1B). *However, the* MLG *is still faster than the latter method by two orders of magnitude for large numbers of nodes and focal nodes* (*e.g.*, $N_f \sim N = 64\text{K}$).

Comparing Figs. 2 and 6, which correspond to the accessing of information on near neighbors, shows that the costs in the two cases are approximately the same. In fact the Type 1 data structure costs somewhat less, in apparent contradiction to ideas previously stated. Because the MLG places the near-neighbor data in the vicinity of each focal node, properly designed parallel hardware should be able to access the near-neighbor data for all nodes simultaneously or at least with appreciable parallelism. For a Type 1 data structure, the indices of near neighbors will be randomly placed in memory and a "gather" operation must be performed. The results of this test point out the importance of the particular hardware. The use of the Cray X-MP is actually optimum for the Type 1 data structure rather than the MLG for two reasons. First, the X-MP has a special hardware gather capability that provides vector speed to the Type 1 random access. Second, the Cray vectorizes over only one index, and longer vector loops are more efficient (faster) than shorter ones. The Type 1 data form singly-indexed, long vectors while the data are triply indexed for a three-dimensional MLG. The MLG vector loop is thus necessarily shorter. The results are still comparable because the vector fetches of data in the MLG test are several times faster than the

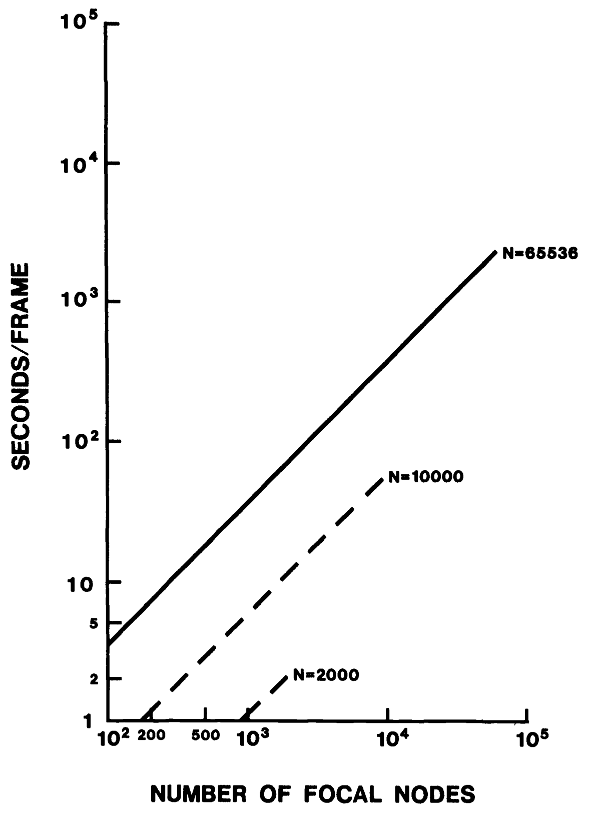


FIG. 5. *Cost per frame of using a conventional (Type* 1) *data structure on the* NRL *Cray* X-MP *computer. Test* 1A: *Identify near neighbors of each focal node by testing to see which nodes fall within a given displacement from the focal node. Results are the same for Test* 1B: *Identify near neighbors of each focal node by testing to see which nodes have spatial coordinates that fall within a prescribed interval about the coordinates of the focal node. The total number of nodes is* 64K. *The dashed lines give predicted results for* $N = 10{,}000$ *and* 2,000.

hardware gather instruction, even though the MLG vector loops are short. Ironically, the Texas Instruments Advanced Scientific Computer (ASC), that preceded the Cray X-MP at NRL was a better model for a parallel processor because the ASC could vectorize three indices simultaneously. This would have improved the speed of data retrieval by a factor of three to four when using the MLG data structure. Note that the data access times were not large enough to affect the comparisons of the overall costs of the MLG and Type 1 data structures.

2.2.2. Tables. Tables 1 and 2 show the results of timing measurements of the MLG data structure for various values of DT and maximum near-neighbor index offsets from the focal nodes. The sorting times are in units of seconds per frame, since the restructuring of the MLG does not depend on the number of focal nodes being processed. The other timings for the MLG and the timings for the Type 1 data structure *do* depend on the number of focal nodes. In molecular dynamics problems, an appreciable fraction of the nodes will be treated as focal nodes. Thus the first three columns must include a multiplicative factor (N_f) of order 10^4 to obtain the time required per frame to perform each task.

The first three columns of Table 1 correspond to writing information on 27 nodes: the 26 near neighbors of index offset 1 and the focal node itself. This is also the case

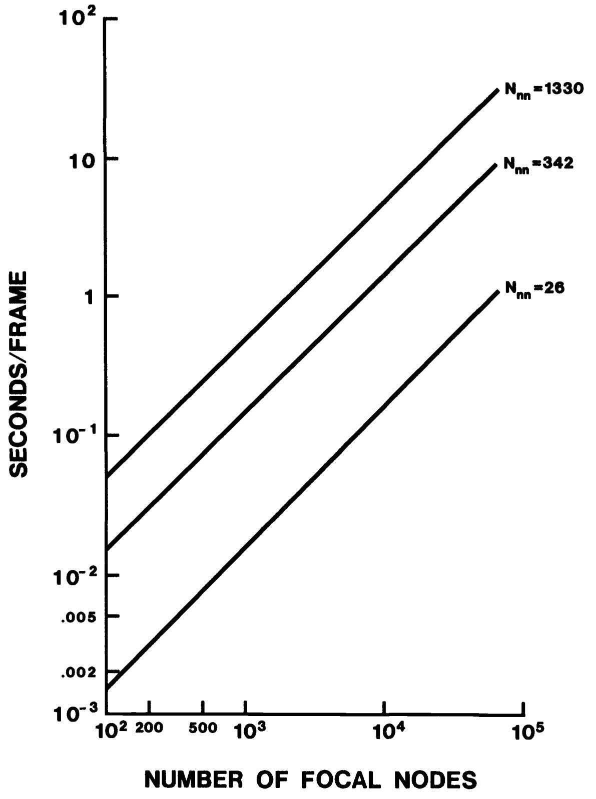


FIG. 6. *Cost per frame of using a conventional (Type 1) data structure on the* NRL *Cray* X-MP *computer. Test 2: Access the data associated with the near neighbors of each focal node and write to a buffer.*

TABLE 1
Times for MLG *data structure tests with index offsets of* 1.*

DT	Test 1†	Test 2	Test 3	Sorting
0.016	$2.93\times10^{-5}N_f$	$5.07\times10^{-5}N_f$	$2.05\times10^{-4}N_f$	8.88×10^{-1}
0.100	$2.91\times10^{-5}N_f$	$5.07\times10^{-5}N_f$	$2.05\times10^{-4}N_f$	1.10×10^{0}
0.600	$2.91\times10^{-5}N_f$	$5.08\times10^{-5}N_f$	$2.04\times10^{-4}N_f$	1.56×10^{0}
3.000	$2.93\times10^{-5}N_f$	$5.07\times10^{-5}N_f$	$2.04\times10^{-4}N_f$	3.05×10^{0}

* Times are in seconds per frame.
† Excludes the sorting time in column four.

TABLE 2
Times for MLG *data structure tests with* DT = 0.6.*

Δ	N_{nn}	Test 1†	Test 2	Test 3	Sorting
1	26	$2.91\times10^{-5}N_f$	$5.08\times10^{-5}N_f$	$2.04\times10^{-4}N_f$	1.56×10^{0}
2	124	$7.46\times10^{-5}N_f$	$1.32\times10^{-4}N_f$	$1.33\times10^{-3}N_f$	1.53×10^{0}
3	342	$1.44\times10^{-4}N_f$	$2.58\times10^{-4}N_f$	$4.35\times10^{-3}N_f$	1.53×10^{0}
5	1330	$3.49\times10^{-4}N_f$	$6.46\times10^{-4}N_f$	$2.05\times10^{-2}N_f$	1.51×10^{0}

* Times are in seconds per frame. The index offset is the same for MLG indices *i*, *j*, and *k*.
† Excludes the sorting time in column four.

in Table 3 for the Type 1 data structure. Note that the first three columns of Table 1 do not change with DT while the sorting time increases as DT increases. This is consistent with our statement in § 2.1 that a larger time interval (DT) between frames will cause more swapping to be performed to maintain the MLG. The largest proportional jump in sorting time occurs on crossing the $DT = 1.0$ threshold. Test 3 is of particular interest because the process of ordering the data according to distance is completed. This requires computing the internode distances and then sorting. Using a vectorized sorting algorithm of order $N \log N$ can reduce this time further. The timing in Test 3 therefore represents an upper bound.

Table 2 shows the effects of increasing the number of near neighbors accessed per focal node. For an offset Δ, the number N_{nn} of near neighbors is $(2\Delta+1)^3-1$. Note that the cost of writing more data goes up, but not linearly, with N_{nn} for Tests 1 and 2. This is because of the longer vector loops involved in accessing the information and writing to the buffer as N_{nn} increases. For Test 3, the time varies approximately as $N_{nn} \log N_{nn}$, since the HEAPSORT algorithm is not vectorized and has approximate computational complexity $M \log M$ for a set of M numbers that must be ordered.

Table 3 shows the results of tests on the Type 1 data structure. Since the nodes have already been sorted on distance in Test 1, Test 3 (find *nearest* neighbors) is automatically satisfied. For the MLG to determine a given number of *nearest* neighbors, the index offset Δ must be sufficiently large to include those *nearest* nodes and an additional sort according to distance must be performed. To carry out any calculation that requires the "nearest neighbors," this extra sort is really unnecessary with the MLG structure, provided that the safety factor (a sufficiently large value of Δ) can be tolerated.

TABLE 3

Times for Type 1 *data structure tests with* $N_{nn} = 26$.*

DT	Test 1†	Test 2	Test 3
0.016	$1.56\times10^{0}N_f$	$1.66\times10^{-5}N_f$	0.0
0.100	$1.57\times10^{0}N_f$	$1.63\times10^{-5}N_f$	0.0
0.600	$1.58\times10^{0}N_f$	$1.63\times10^{-5}N_f$	0.0
3.000	$1.57\times10^{0}N_f$	$1.62\times10^{-5}N_f$	0.0

* Times are in seconds per frame.
† Includes HEAPSORT.

Table 3 shows that the cost of Test 1 when processing N_f focal nodes for the Type 1 data structure is

$$C_{\text{Test 1}} = N_f \times 1.6 \times [(N \log N)/(65536 \log 65536)] \text{ sec/frame.} \tag{6}$$

Equation (6) includes the scaling of HEAPSORT for N nodes. For $N = 64K$, Table 2 (column one plus column four) for the MLG and (6) tell us that the Type 1 data structure will require between three and four orders of magnitude more computing time for identifying near neighbors when using HEAPSORT than will the MLG data structure. If the method of Test 1A or 1B to identify near neighbors is used, the Type 1 data structure will still cost two orders of magnitude more for reasonable numbers of focal nodes ($N_f \sim N$). The reason is that, for each focal node, 64K distances must be computed and then sorted (Test 1) or partitioned (Tests 1A and 1B). Thus, the cost of identifying near neighbors scales roughly as $N \times N_f$. For $N_f = N$, the cost scales approximately as N^2. Maintaining the MLG requires only one global sort, and identifying near neighbors is thus of order N_f.

Table 4 shows how the Type 1 data structure performs as a function of the number of near neighbors for which data must be accessed. Note that the ordering of nodes according to distance from each focal node (Test 1) does not change with N_{nn}. However, the cost of accessing the data (Test 2) scales roughly with N_{nn}, as we might expect.

Table 5 shows how the cost of identifying near neighbors varies with the number of near neighbors desired in the case of Tests 1A and 1B. Test 1A finds those nodes that are within some prescribed spatial displacement of each focal node, while Test 1B finds those nodes whose three spatial coordinates fall within some coordinate interval containing a given focal node. As in the case of Test 1, the timings do not depend on either DT or N_{nn}. In fact, the number of near neighbors that are found in Tests 1A and 1B will depend on the particular configuration of nodes found at a given time and the value of the radius and the coordinate interval, respectively, that have been chosen to define which nodes are "near" neighbors.

3. Monotonic logical grid versus "Type 2" data structures. This section briefly describes another class of data structures applicable to collections of moving nodes with interactions between near neighbors. These data structures, denoted by "Type 2" (T2), result in a computational complexity of order N for computing interactions, as does the MLG. However, unlike the MLG, the T2 structures require extra memory to carry information in the form of pointers that indicate which nodes are near neighbors. This makes the T2 structures significantly more complex than the MLG, which simply rearranges memory (index space) to reflect near-neighbor information. The T2 pointers often appear in the form of linked-list variables that relate spatial groupings of nodes.

TABLE 4
Times for Type 1 data structure tests with DT = 0.6.*

N_{nn}	Test 1†	Test 2	Test 3
26	$1.58\times10^0 N_f$	$1.63\times10^{-5} N_f$	0.0
124	$1.59\times10^0 N_f$	$5.47\times10^{-5} N_f$	0.0
342	$1.57\times10^0 N_f$	$1.40\times10^{-4} N_f$	0.0
1330	$1.59\times10^0 N_f$	$5.22\times10^{-4} N_f$	0.0

* Times are in seconds per frame.
† Includes HEAPSORT.

TABLE 5
Times for alternative Type 1 data structure tests with DT = 0.6.*

N_{nn}	Test 1A†	Test 1B‡
30-70	$3.63\times10^{-2}N_f$	$3.57\times10^{-2}N_f$
100-300	$3.63\times10^{-2}N_f$	
500	$3.64\times10^{-2} N_f$	$3.79\times10^{-2} N_f$

* Times are in seconds per frame.
† Find nodes within a prescribed distance of a given focal node.
‡ Find nodes with coordinates near those of a given focal node.

This feature could render them unvectorizable. As an example, we will describe the method of Hockney and Eastwood [3]. Another T2 method is that of Appel [1], which uses a hierarchy of clusters to represent near-neighbor relationships. Both of these papers emphasize the methods of computing Coulombic or gravitational near-neighbor interactions as much as the actual construction of the data structures. In addition to the T2 structures designed for many-body dynamics, some of the more general near-neighbor search algorithms are of interest here. The last section below briefly describes the k nearest-neighbor finding algorithm of Kim and Park [4], which bears some resemblance to the Monotonic Logical Grid.

3.1. The method of Hockney and Eastwood. The data structure of Hockney and Eastwood, denoted by "HET2," covers the physical space with a set of identical cubical cells, the length of a side being determined by the range of the near-neighbors interaction. Each cell has a "head of chain" (HOC) variable that is one word in length, and each moving node has a linked-list (LL) variable in addition to the other associated data (e.g., position, velocity). The associated data are indexed by the particle ID as in the Type 1 data structure. The HOC and LL variables form chains that identify all of the node IDs associated with each spatial cell. In this way, the scheme maintains coarse information on near neighbors: the near neighbors of a node are the other nodes in the same cell plus those in the adjoining cells.

A simple example illustrates how this might be done. Suppose that the number of nodes N is 100 and that at time t the nodes with ID = 10, 30, and 50 lie in cell number 12. At each time step or frame, the algorithm cycles through the position data of all of the nodes in order of ID value. Before the loop through node IDs is started, all cells have an HOC of zero. For each successive node, beginning with ID = 1, three divide operations are performed to determine which cell contains the node. For node ID = 10, the calculation shows that node 10 is in cell number 12. The algorithm sets LL(10) = 0, the value of HOC for cell 12 (that is, HOC(12)). The code sets HOC(12) = 10, the ID of the first node found to lie in cell 12. At node number (ID =) 30, the calculation again shows that this node is in cell 12. The code sets LL(30) = HOC(12) = 10, the ID of the first node found to be in cell 12. The code than sets HOC(12) = 30, the ID of the second node found to lie in cell 12. At ID = 50, the algorithm finds the third node occupying cell 12. The algorithm sets LL(50) = HOC(12) = 30 and HOC(12) = 50. After the loop over IDs is finished, the code can find which nodes are in cell 12 by reading HOC(12) to obtain ID = 50, then reading LL(50) to obtain ID = 30, and finally reading LL(30) to obtain ID = 10. Because LL(10) = 0, only those three nodes lie in cell 12.

After this procedure is performed, the "near neighbors" of a given node are then all nodes found in the same cell plus the nodes in some set of the neighboring cells. As in the case of the MLG, this is only coarse information on relative location, and further distance calculations and sorting may be required. The details of updating or recomputing the linked lists at each timestep will vary with user and application.

We need not perform test calculations to observe several advantages of the MLG over this class of T2 schemes:

(1) Because the MLG manipulates computer memory directly rather than external mapping of the memory (HOC and LL variables), the use of memory is more efficient. The HET2 scheme requires extra memory in the form of one HOC per cell and one LL variable per node, and an appreciable number of the cells could be empty.

(2) Depending on the problem, the optimum sizes and numbers of cells might vary significantly over time. Furthermore, the cells must have constant or regularly

varying size in each dimension independently of the others. The partitioning of space might not, therefore, be very efficient.

(3) The computation of near-neighbor interactions using the HET2 structure is not particularly vectorizable and cannot take advantage of a hardware gather capability such as that on the NRL Cray X-MP. Replacing the linked-list concept by separate list vectors for each cell would permit us to take advantage of the hardware gather capability. This would drastically increase the requirements for memory or at least the difficulty in allocating memory efficiently. The recent appearance of new multiple-instruction-stream, multiple-data-stream (MIMD) and single-instruction-stream, multiple-data-stream (SIMD) computers might permit parallelization of these algorithms, bypassing the vectorization issue.

(4) Local restructuring of the linked-lists from frame to frame is not vectorizable in the above prescription and may not permit vectorization on a Cray, for example. The actual expense of constructing the linked lists at each timestep or frame, however, might still be small.

(5) The T2 algorithm is more complex and cumbersome than the MLG algorithm and thus more susceptible to programming errors. Moreover, extension of T2 to four-dimensional (space-time) problems or even higher dimensions would be far less transparent than for the MLG.

The extension of the MLG data structure to higher dimensions could be quite important in some calculations. Attributes distinguishing nodes from each other would, in fact, determine how various nodes participate in the calculation. The MLG data structure could include extra dimensions corresponding to these attributes, so that near neighbors in index space would be spatial near neighbors *with similar attribute values.* We can also envision using a number of relational operators in addition to "less than or equal to" ($\leqq$), which has been used to construct the three-dimensional MLG data structure considered in this paper.

3.2. The method of Kim and Park. One class of algorithms that is relevant to our discussion but that has not been designed particularly for many-body calculations is that of k nearest neighbor searches. Kim and Park [4] have recently presented an algorithm of this type based on an ordered partition. The method of Kim and Park, denoted by "KP," has similarities to both the MLG and the Type 2 data structures designed for many-body dynamics. The statistical analysis presented by KP for their new algorithm thus sheds some light on the MLG algorithm and perhaps other Type 2 methods.

As an example relevant to our paper, consider a set S of N objects, each having a set of coordinates (x, y, z) specifying its spatial location. For a given point with coordinates (x', y', z'), KP have devised a two-step procedure for finding the k objects in S that are nearest to these coordinates. The first step consists of constructing a search tree through ordering and partitioning the N sets of coordinates, independently of the focal coordinates (x', y', z'). The second step involves a search that is quite fast because of the manner in which the tree was constructed.

This method bears some similarities to the MLG approach (Appendix I). The ordering and partitioning at the lth level of the search tree depends only on the values of the lth coordinate for the N objects in S. Furthermore, only one search tree need be constructed for a given configuration of objects (corresponding to a given timestep in a molecular dynamics simulation, for example). The nearest objects within S to any value of (x', y', z') may be accessed using the same search tree. In the case of the MLG, the values of the lth MLG index assigned to the members of S depend only on the respective values of the lth spatial coordinate. In addition, only one global MLG restructuring step is required for a given configuration or timestep. The latter feature is also true of other Type 2 data structures.

Significant differences between the MLG and the KP algorithm also exist. For many-body dynamics calculations, the KP algorithm appears to require the construction of a new search tree from scratch at each timestep, while MLG restructuring requires only local swaps of data in memory. The possibility exists that a similar local restructuring of the search tree might be devised. A second difference is that the KP algorithm requires N_f searches to find the neighbors of N_f focal nodes in a many-body simulation. The MLG requires no searching, since access of near neighbors is automatic. On the other hand, the reader should recognize that, for the fast k nearest neighbor finding algorithm of KP, the focal coordinates (x', y', z') need not correspond to some member of S. The MLG algorithm, in contrast, applies most directly to near-neighbor relationships or interactions among the objects within S. The KP algorithm is thus more general in this regard, although further development of the MLG may alter the situation.

The Kim–Park Algorithm is like other Type 2 data structures in that the storage of data is static. Pointers map the object data to nodes in the search tree, playing the same role as the linked lists in the Type 2 data structure of Hockney and Eastwood, for example. The accessing of data on near neighbors then requires a "gather" operation. Another similarity is the difficulty of vectorization of near-neighbor calculations on a Cray using the search tree or, alternatively, the need for significant amounts of memory to construct arrays of data on near neighbors to permit vectorization of interaction calculations.

Some of the above disadvantages might not be as limiting as they first appear. New MIMD and SIMD computers should provide parallelization of near-neighbor calculations using the KP algorithm, bypassing vectorization problems. The possibility also exists that the storage of data in computer memory might be rearranged according to the structure of the search tree, thus adopting the philosophy behind the construction of the MLG.

4. Summary. We have investigated the relationships of the MLG data structure to two standard data structures and compared the different algorithms for several tasks:

(1) Identifying the near neighbors of a set of focal nodes within a set of noninteracting nodes that move randomly;

(2) Retrieving information on the near neighbors that were identified in (1); and

(3) Ordering the information retrieved on the near neighbors. The order corresponded to the distances of near neighbors from the respective focal nodes.

In many-body calculations, a large fraction of the total set of nodes will have to be treated as focal nodes throughout the evolution of the system. Thus the access of data on near neighbors will scale as N^2—the so-called "combinatorial explosion"—for most conventional data structures. Such conventional structures generally fall in the Type 1 category considered above. Even for the clever Type 2 approaches considered in § 3, implementation and performance for many-body dynamics on a given parallel architecture might fall short of the MLG. In the latter case, the source of the advantage is that the MLG restructures data in computer memory directly—without an explicit apparatus such as linked lists.

Two parameters that determine the performance of the MLG are the dimensionless frame time DT and the maximum index offset Δ. The value of DT will determine how much work will be required to restructure the MLG at each timestep. If the value of DT is less than one, the MLG is easy to maintain. While the required index offset depends on the density of nodes and the desired number of *nearest* neighbors, offsets larger than four or five should seldom be necessary.

The MLG approach has several strengths: a requirement for only one reordering process per frame, order N scaling of calculations of near-neighbor interactions and associations, ready implementation on vector or highly parallel computers, and direct restructuring of memory, eliminating the need for an expensive, partly scalar apparatus (e.g., linked lists) for the maintenance of near-neighbor identification data. As an interesting side issue, computer hardware can directly impact the comparisons of data structures. In this case, the Cray X-MP hardware gather capability permitted the test code to retrieve near-neighbor data somewhat more quickly for the Type 1 data structure than for the MLG data structure, when both were programmed in FORTRAN. This was not, however, a significant factor in the comparison.

Appendix. Constructing a Monotonic Logical Grid—An example. Begin with a collection of 16 objects randomly distributed in space as shown in Fig. A1. We will organize these in a section of computer memory so that near neighbors in physical space will be near neighbors in index space. This indexing scheme or rule for the data organization in memory is an example of a Monotonic Logical Grid (MLG).

Step 1. Give each object x and y coordinates in two-dimensional space relative to some origin. This has been done in Fig. A1.

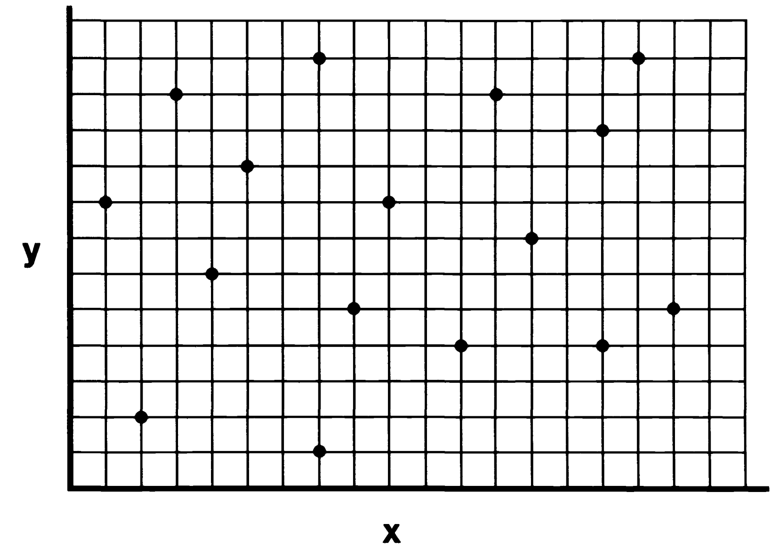


FIG. A1

Step 2. Decide on the "type" of MLG one wants. This decision is expressed in terms of a rule for constructing the grid. The rule may, for example, be stated as follows:

Rule. Choose a computer memory location or index (i, j) for each object such that

$$x(i,j) \leqq x(i+1,j), \qquad y(i,j) \leqq y(i,j+1)$$

where i and j run from one to four.

Step 3. Implement the rule as follows, remembering that the MLG chosen here will occupy a 4×4 index space in memory:

(A) Order all 16 objects according to increasing y-coordinate, that is, from lowest to highest y-value. In Fig. A2 these are labeled A through P.

(B) Give the objects with the four lowest y-values (that is, objects A, B, C, and D) a j index of one. Give those with the next four lowest y-values (E, F, G, and H) a j-index of two, and so on. Figure A3 shows each set of four connected with straight lines.

(C) Now assign an i-index to each object in a given set of four with the same j-index according to the selected MLG rule. This assignment is in Fig. A4. Note that those objects with the same i-index are connected with a line.

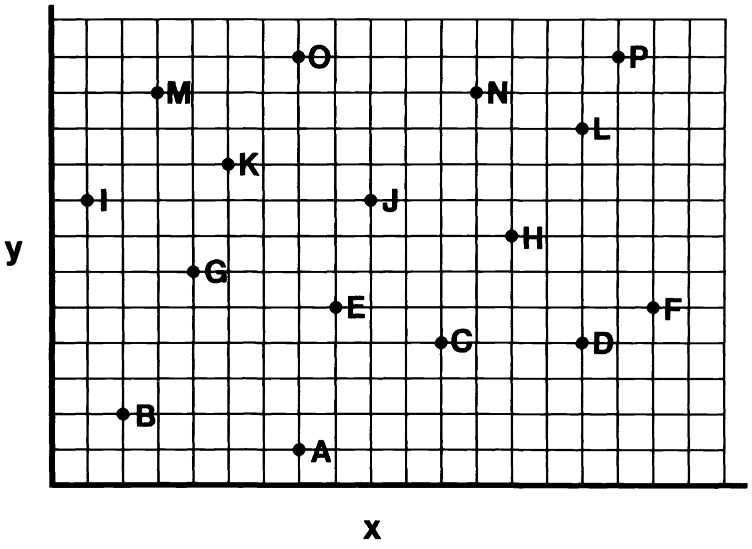


FIG. A2

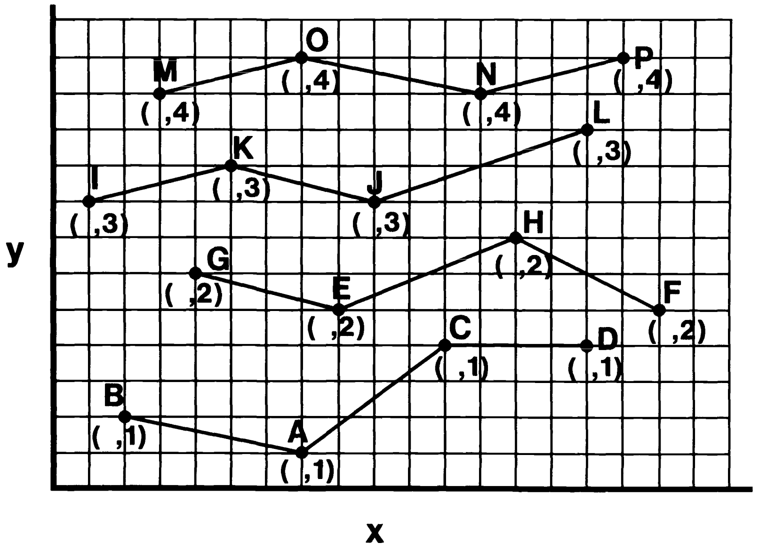


FIG. A3

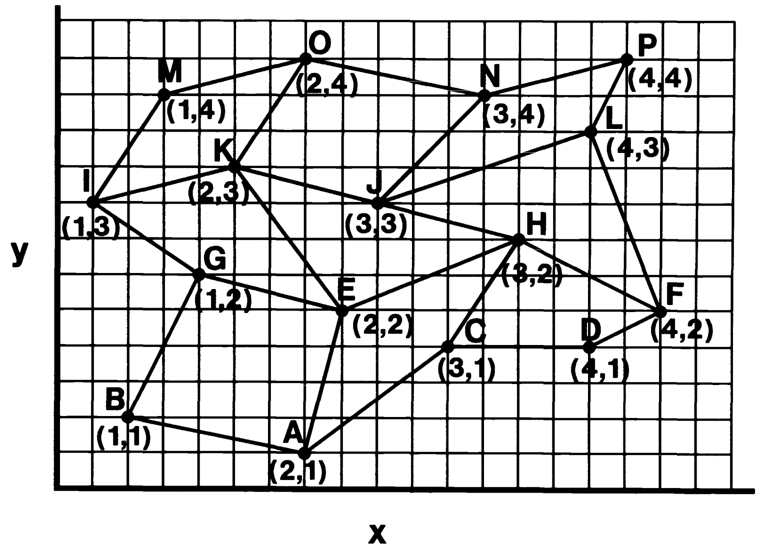


FIG. A4

The two-dimensional MLG in index space looks like Fig. A5. The letters in Fig. A5 match the letters labeling specific nodes on the previous figures. Nodes (letters) that are adjacent in memory (index space) are connected by lines in Figs. A3 and A4. Thus adjacency of the node data in memory corresponds to geometric adjacency of the nodes.

The eight near neighbors of point *H* in index space are points *A*, *C*, *D*, *E*, *F*, *K*, *J*, and *L* in real space as shown in Fig. A6. The dashed line in Fig. A6 encloses the near neighbors of *H* in index space (compare with Fig. A5).

Acknowledgments. We appreciate many helpful discussions with Mr. M. S. McBurnett and Dr. S. Jajodia of the Naval Research Laboratory Information Technology Division.

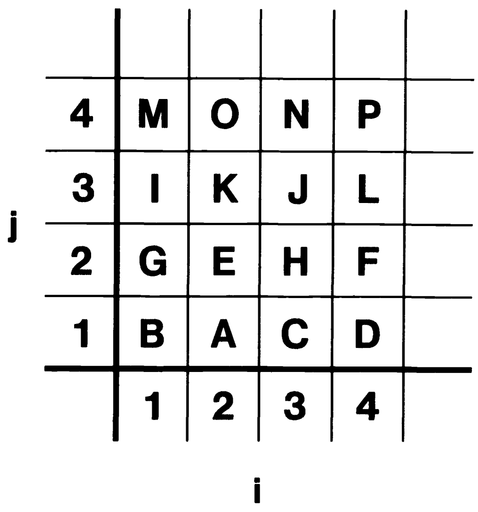


FIG. A5

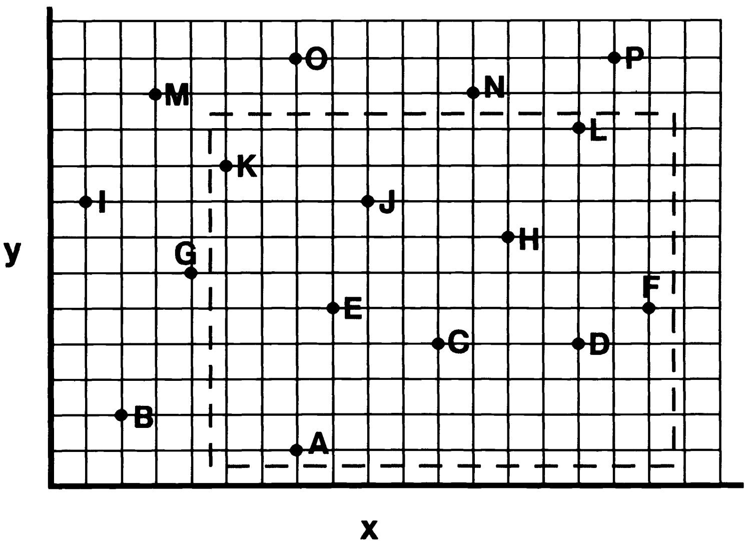


FIG. A6

REFERENCES

[1] A. W. APPEL, *An efficient program for many-body simulation*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 85–103.

[2] J. P. BORIS, *A vectorized "near neighbors" algorithm of order N using a monotonic logical grid*, J. Comput. Phys., 66 (1986), pp. 1–20.

[3] R. W. HOCKNEY AND J. W. EASTWOOD, *Computer Simulation Using Particles*, McGraw-Hill, New York, 1981, Chap. 8, pp. 267–309.

[4] B. S. KIM AND S. B. PARK, *A fast k nearest neighbor finding algorithm based on the ordered partition*, IEEE Trans. Pattern Analysis and Machine Intelligence, 8 (1986), pp. 761–766.

[5] S. G. LAMBRAKOS AND J. P. BORIS, *Geometric properties of the monotonic Lagrangian grid algorithm for near-neighbor calculations*, J. Comput. Phys., 73 (1987), pp. 183–202.

[6] S. G. LAMBRAKOS, J. P. BORIS, R. GUIRGUIS, M. PAGE, AND E. S. ORAN, *Molecular dynamics simulation of* $(N_2)_2$ *formation using the monotonic Lagrangian grid*, J. Chem. Phys., (1989), pp. 4473–4481.

[7] A. NIJENHUIS AND H. S. WILF, *Combinatorial Algorithms for Computers and Calculators*, Academic Press, New York, 1978, p. 140.

[8] D. B. REID, *An algorithm for tracking multiple targets*, IEEE Trans. Automat. Control, 24 (1979), pp. 843–854.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 2, pp. 389–397, March 1990

A NOTE ON THE USE OF SYMMETRIC LINE GAUSS–SEIDEL FOR THE STEADY UPWIND DIFFERENCED EULER EQUATIONS*

WIM A. MULDER†

Abstract. Symmetric Line Gauss–Seidel (SLGS) relaxation, when used to compute steady solutions to the upwind differenced Euler equations of gas dynamics, is shown to be unstable. The instability occurs for the long waves. If SLGS is used in a multigrid scheme, stability is restored. However, the use of an unstable relaxation scheme will not provide a robust multigrid code. Damped Symmetric Point Gauss–Seidel relaxation is stable and provides similar multigrid convergence rates at much lower cost. However, it fails if the flow is aligned with the grid over a substantial part of the computational domain. Damped Alternating Direction Line Jacobi relaxation can overcome this problem.

Key words. line relaxation, stability, multigrid method, steady Euler equations

AMS(MOS) subject classifications. 35L65, 65N20, 76N15

1. Introduction.

The Euler equations that describe the flow of an inviscid compressible gas can be integrated in time by means of an implicit method. This is desired if the flow displays features on different scales, or if the steady state has to be computed efficiently. The implicit formulation gives rise to a large sparse system of linear equations, which can be solved by factorization methods such as the Alternating Direction Implicit method [4] and Approximate Factorization [1]. However, if the spatial discretization is obtained by upwind differencing, the implicit system is diagonally dominant and can be solved more efficiently by classical relaxation methods. This was pointed out and explored by van Leer and myself [15], and, independently, by Chakravarthy [3].

In [15] we found that one particular relaxation method, Symmetric Line Gauss–Seidel (SLGS), did not converge as fast as expected, and sometimes did not converge at all. This was thought to be caused by the numerical boundary conditions. The same problem was encountered in [13].

Here it is shown that the convergence problem is due to the intrinsic instability of the SLGS scheme. In §2, the classical von Neumann stability analysis is carried out on the system of linearized Euler equations in two dimensions. As the Fourier modes used in the analysis are not the proper eigenfunctions for Gauss–Seidel relaxation, some numerical experiments were carried out with the nonlinear equations (§3).

The instability occurs for the long waves. If SLGS is applied as a relaxation scheme in a multigrid code, the corrections from coarser grids are sufficient to suppress the instability. This is shown in §4. A multigrid scheme based on SLGS relaxation has already been used in [8]. However, damped Symmetric Point Gauss–Seidel (SGS) provides similar convergence rates at a lower cost, and damped Alternating Direction Line Jacobi has much better convergence factors.

The main results are summarized in §5.

*Received by the editors August 29, 1988; accepted for publication (in revised form) March 27, 1989. This work was supported by the Center for Large Scale Scientific Computing (CLaSSiC) Project at Stanford University, under Office of Naval Research contract N00014-82-K-0335.

†Department of Computer Science, Stanford University, Stanford, California 94305-2140. Present address, Koninklijke/Shell Exploratie en Produktie Laboratorium, Postbus 60, 2280 AB Rijswijk, the Netherlands.

2. Stability analysis. The flow of an ideal inviscid compressible gas can be described by the Euler equations. In conservation form, these are given by

$$\frac{\partial w}{\partial t} + \frac{\partial f}{\partial x} + \frac{\partial g}{\partial y} = 0. \tag{2.1a}$$

The vector of states w and the fluxes f and g are

$$w = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix}, \quad f = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho u H \end{pmatrix}, \quad g = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho v H \end{pmatrix}. \tag{2.1b}$$

The density of the gas is denoted by ρ. The x- and y-component of the velocity are u and v, respectively. The energy E, total enthalpy H, pressure p, and sound speed c are related by

$$E = \frac{1}{(\gamma-1)}\frac{p}{\rho} + \frac{1}{2}(u^2+v^2), \qquad H = E + \frac{p}{\rho}, \qquad c^2 = \gamma\frac{p}{\rho}. \tag{2.2}$$

A nonconservative form of (2.1) is

$$\frac{\partial w'}{\partial t} + A\frac{\partial w'}{\partial x} + B\frac{\partial w'}{\partial y} = 0, \tag{2.3a}$$

where

$$A = \begin{pmatrix} u & 0 & c & 0 \\ 0 & u & 0 & 0 \\ c & 0 & u & 0 \\ 0 & 0 & 0 & u \end{pmatrix}, \quad B = \begin{pmatrix} v & 0 & 0 & 0 \\ 0 & v & c & 0 \\ 0 & c & v & 0 \\ 0 & 0 & 0 & v \end{pmatrix}, \quad \delta w' = \begin{pmatrix} \delta u \\ \delta v \\ \delta p/(\rho c) \\ \delta S \end{pmatrix}. \tag{2.3b}$$

Here $S = \log(p/\rho^\gamma)$ is the specific entropy.

The stability analysis will be carried out for the discretization of the linear residual operator

$$L = A\frac{\partial}{\partial x} + B\frac{\partial}{\partial y}, \tag{2.4}$$

with constant coefficients and periodic boundary conditions. The operator is discretized in space by upwind differencing. This is accomplished as follows. The matrix A is diagonalized by Q_1, according to

$$A = Q_1\Lambda_1 Q_1^{-1}, \quad \Lambda_1 = \begin{pmatrix} u-c & & & \bigcirc \\ & u & & \\ & & u & \\ \bigcirc & & & u+c \end{pmatrix}, \quad Q_1 = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{2.5a}$$

For B we have

$$B = Q_2\Lambda_2 Q_2^{-1}, \quad \Lambda_2 = \begin{pmatrix} v-c & & & \bigcirc \\ & v & & \\ & & v & \\ \bigcirc & & & v+c \end{pmatrix}, \quad Q_2 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ -1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{2.5b}$$

To obtain an upwind scheme, the matrix Λ_k ($k = 1, 2$) is split into Λ_k^+ and Λ_k^-, which contain the positive and negative elements of Λ_k, respectively. This implies

$$\Lambda_k^+ + \Lambda_k^- = \Lambda_k, \qquad \Lambda_k^+ - \Lambda_k^- = |\Lambda_k|. \tag{2.6}$$

Now define

$$A^\pm \equiv Q_1\Lambda_1^\pm Q_1^{-1}, \qquad B^\pm \equiv Q_2\Lambda_2^\pm Q_2^{-1}. \tag{2.7}$$

It follows that

$$\begin{aligned} A &= A^+ + A^-, \qquad |A| \equiv Q_1|\Lambda_1|Q_1^{-1} = A^+ - A^-;\\ B &= B^+ + B^-, \qquad |B| \equiv Q_2|\Lambda_2|Q_2^{-1} = B^+ - B^-. \end{aligned} \tag{2.8}$$

The discrete linear residual operator becomes

$$L^h \equiv \frac{1}{h_x}[A^+(1 - T_x^{-1}) + A^-(T_x - 1)] + \frac{1}{h_y}[B^+(1 - T_y^{-1}) + B^-(T_y - 1)]. \tag{2.9}$$

Here the shift operators are defined by $T_x w_{k_1,k_2} \equiv w_{k_1+1,k_2}$, $T_y w_{k_1,k_2} \equiv w_{k_1,k_2+1}$. For simplicity, the grid is assumed to be uniform ($h_x = h_y = h$).

For the stability analysis we consider the usual Fourier modes of the form

$$\exp\left[-i\left(k_1\theta_x + k_2\theta_y\right)\right], \tag{2.10a}$$

where k_1 and k_2 are spatial indices on a $N_1 \times N_2$ grid. The frequencies for a grid of the same size are

$$\theta_x = 2\pi\frac{l_1}{N_1}, \quad \theta_y = 2\pi\frac{l_2}{N_2}, \quad l_1 = 0, \cdots, N_1 - 1, \quad l_2 = 0, \cdots, N_2 - 1. \tag{2.10b}$$

The Fourier transforms of the shift-operators T_x and T_y are

$$\hat{T}_x \equiv \exp(i\theta_x), \quad \hat{T}_y \equiv \exp(i\theta_y), \quad 0 \le \theta_x < 2\pi, \quad 0 \le \theta_y < 2\pi. \tag{2.11}$$

The relaxation operator for SLGS, with the line in the y-direction, is the product of a forward sweep

$$\hat{G}_1 = I - \hat{M}_1^{-1}\hat{L}^h, \qquad \hat{M}_1 = \hat{L}^h - \frac{1}{h}A^-\hat{T}_x, \tag{2.12a}$$

and a backward sweep

$$\hat{G}_2 = I - \hat{M}_2^{-1}\hat{L}^h, \qquad \hat{M}_2 = \hat{L}^h + \frac{1}{h}A^+\hat{T}_x^{-1}, \tag{2.12b}$$

resulting in

$$\hat{G}^{\mathrm{SLGS}} = \hat{G}_2\hat{G}_1. \tag{2.12c}$$

Here I denotes a 4×4 identity matrix. If $\hat{M}_1$ or $\hat{M}_2$ is singular, the pseudo-inverse should be used. If these matrices are nonsingular, then we obtain

$$\hat{G}^{\mathrm{SLGS}} = \hat{M}_2^{-1}A^+\hat{M}_1^{-1}A^-. \tag{2.12d}$$

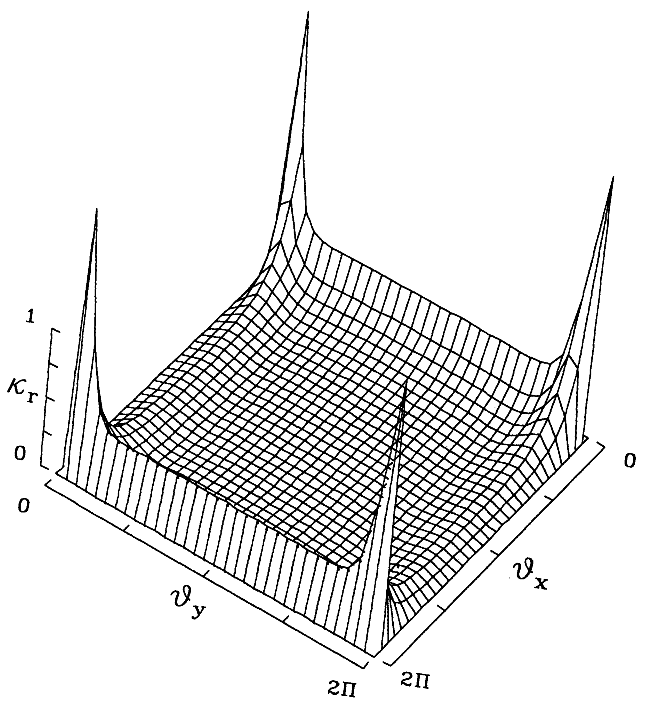


FIG. 1. *Amplification factor of* SLGS *relaxation for* $u/c = 0.8$ *and* $v/c = 0$. *The figure corresponds to a* 32×32 *grid.*

This shows that SLGS is an exact solver if $|u| \geq c$, because either A^+ or A^- vanishes in that case. The fourth equation of (2.3), which describes the convection of entropy along streamlines, is also solved exactly, for all u and v.

The stability of this scheme is investigated by computing the amplification factor of the residual

$$\text{(2.13)} \qquad \overline{\kappa}_r \equiv \max_{\theta_x,\theta_y} \kappa_r(\theta_x,\theta_y), \qquad \kappa_r(\theta_x,\theta_y) \equiv \rho\Big(\hat{L}^h \hat{G}^{\text{SLGS}}(\hat{L}^h)^\dagger\Big).$$

Here $\rho(\cdot)$ denotes the spectral radius. The operator $\hat{L}^h$ can be singular [9, Lem. 3.1], and the waves for which it is singular are obviously not damped or amplified. To exclude these waves, the operator $\hat{L}^h$ and its pseudo-inverse are included in the definition of κ_r. The value of $\overline{\kappa}_r$ will depend on the velocities u/c and v/c, and on the grid-size $N_1 \times N_2$.

Figure 1 shows $\kappa_r(\theta_x, \theta_y)$ for $u/c = 0.8$ and $v/c = 0$. Here $N_1 = N_2 = 32$. The instability is clearly visible. It occurs for the long waves, at $|\theta_x| = |\theta_y| = 2\pi/32$. For the given u/c and v/c, the instability does not show up on a 16×16 grid, whereas it becomes worse on finer grids.

Similar instabilities occur for most values of $|u/c| < 1$ and $|v/c| \leq 1$, in some cases on grids coarser than in the example of Fig. 1.

3. Numerical experiments.

It is well known that Fourier modes are not the proper eigenfunctions for Gauss–Seidel relaxation. Therefore, the validity of the above

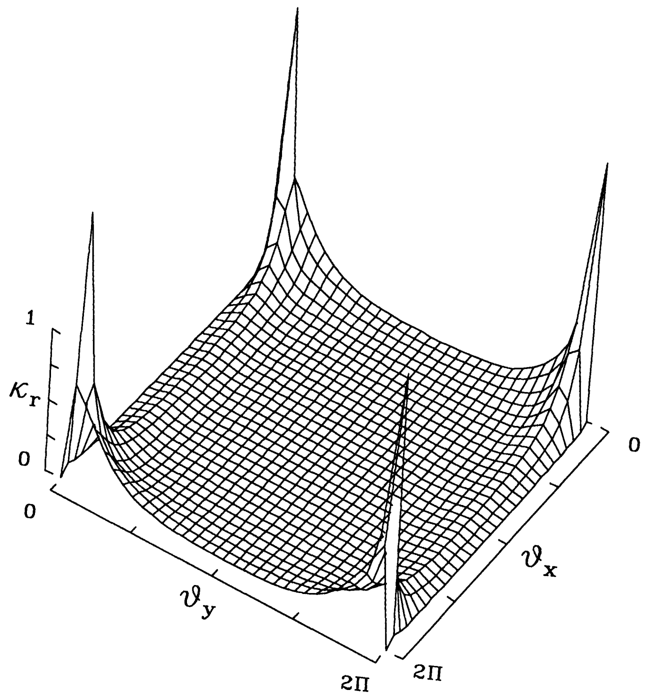


FIG. 2. *As Fig.* 1, *but for van Leer's flux-vector splitting.*

analysis may be questioned, especially for the longer waves. In this section the instability will be investigated by numerical experiments on the system of nonlinear Euler equations (2.1).

For the upwind differencing, van Leer's flux-vector splitting (FVS) [14] is used as an approximate Riemann-solver. This scheme gives rise to matrices $A^{\pm} \equiv df^{\pm}/dw$ and $B^{\pm} \equiv dg^{\pm}/dw$, which are different from those in (2.7). Therefore, the stability properties of this scheme will be different from those predicted in the previous section, but not by too much. Figure 2 shows the amplification factor for FVS, using the same parameters as in Fig. 1.

As a test problem, flow through a straight channel is considered. The grid is square and uniform. There are hard walls on the lower and upper sides. Boundary conditions at the walls are implemented by mirror cells that contain reflected states. Characteristic boundary conditions are used at the inlet and outlet. In principle, overspecification can be used because the Riemann-solver takes care of the appropriate switching between incoming and outgoing characteristics. However, because FVS is not a very good approximate Riemann-solver, the use of characteristic boundary conditions is recommended.

The free-stream values are chosen to be

$$\rho_\infty = 1, \qquad u_\infty = 0.8, \qquad v_\infty = 0, \qquad c_\infty = 1, \tag{3.1}$$

whereas the gas-constant $\gamma = 1.4$. As initial conditions, we take the free-stream values and add random noise with an amplitude of 10^{-5}. The steady state is given by the free-stream values.

TABLE 1

Amplification factors for flux-vector splitting on grids of various sizes, with $u_\infty/c_\infty = 0.8$ and $v_\infty/c_\infty = 0.0$. The result of the Fourier stability analysis is denoted by $\overline{\kappa}_r$. The other values are determined from numerical experiments on the full system of nonlinear Euler equations.

N	$\overline{\kappa}_r$	Observed
16×16	0.594	0.80
32×32	1.853	1.45
64×64	4.096	2.55

Table 1 shows the predicted and observed amplification factors. There is a clear qualitative agreement. Inspection of the difference between the nonconverged solution and the numerical steady state confirms the long-wave instability.

4. Multigrid. The instability of SLGS occurs for the long waves. It is therefore expected that the combination of SLGS and the multigrid technique will provide a stable scheme. The analysis of multigrid convergence for the linearized Euler equations with constant coefficients is presented in detail in [9]. The multigrid convergence factor $\overline{\lambda}_r$ of a given relaxation scheme is estimated by considering two grids, a fine and a coarse. The number of cells on the coarse grid is one-fourth of that on the fine. For the restriction to the coarser grid, simple averaging is used. Piecewise constant interpolation is applied for the prolongation back to the fine grid (cf. [7]). In the analysis, it is assumed that the coarse-grid equations are solved exactly. The combined result of the coarse-grid correction and the relaxation scheme is described by $\overline{\lambda}_r$. This two-level multigrid convergence factor gives a reasonable estimate of the convergence speed when many coarser grids are used.

The result of the so-called two-level analysis is shown in Fig. 3. The instability has disappeared. The overall convergence rate is good, except near the singularities of the (linear) residual. The slow convergence around $v = 0$ occurs when the flow is aligned with the grid over a substantial part of the computational domain. To overcome the problem of strong alignment [2], one might consider a relaxation scheme that consists of a SLGS step with the line in the x-direction, followed by SLGS with the line in the y-direction. This will be called Alternating Direction SLGS. ADSLGS turns out to be stable in a larger region of the $(u/c, v/c)$-plane. However, the instability persists for some parameters and is so severe that it does not disappear in a multigrid scheme.

A scheme that suffers from strong alignment in a similar way as SLGS is damped Symmetric Point Gauss–Seidel (SGS) [9]. It is stable as a single-grid scheme and much cheaper than its undamped line variant, and is therefore to be preferred. If one wants a uniformly good convergence rate, a multigrid scheme that employs damped Alternating Direction Line Jacobi (ADLJ) can be used. The linear two-level analysis predicts a multigrid convergence rate $\overline{\lambda}_r(u/c, v/c) \leq 0.526$ for this scheme. The relaxation operator is described by the product of line relaxation in the y-direction, namely,

$$\hat{G}_1 = I - \hat{H}_1^{-1}\hat{L}^h, \qquad \hat{H}_1 = \hat{L}^h + \frac{1}{h}[A^+(\hat{T}_x^{-1} + 1) - A^-(\hat{T}_x + 1)], \tag{4.1a}$$

and line relaxation in the x-direction:

$$\hat{G}_2 = I - \hat{H}_2^{-1}\hat{L}^h, \qquad \hat{H}_2 = \hat{L}^h + \frac{1}{h}[B^+(\hat{T}_y^{-1} + 1) - B^-(\hat{T}_y + 1)]. \tag{4.1b}$$

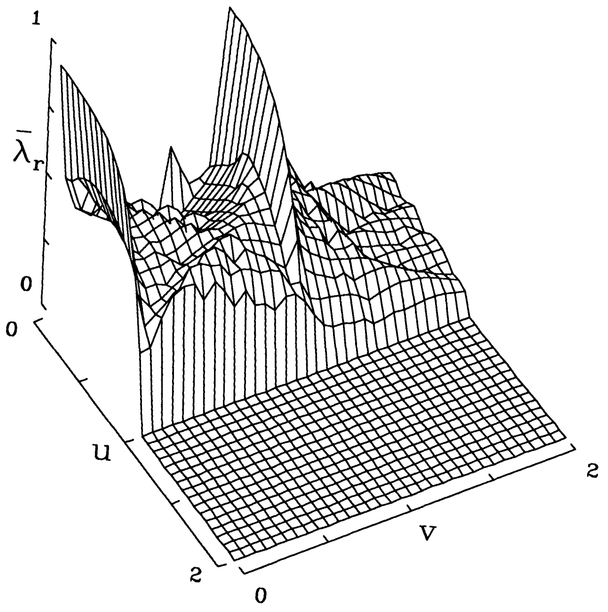


FIG. 3. *Multigrid convergence factor for* SLGS *relaxation as a function of u and v ($c = 1$). Each point is computed for a* 64×64 *grid.*

Note that the relaxation operators H_1 and H_2 are obtained by selecting the main-diagonal (in terms of blocks) and two off-diagonals (in one direction) from the residual operator. The damping is obtained by subtracting the two other off-diagonals in the direction perpendicular to the line from the main-diagonal. Figure 4 shows the two-level multigrid convergence rate for damped ADLJ relaxation.

5. Conclusions.

SLGS relaxation is unstable if used for the implicit upwind differenced Euler equations. The instability occurs for the long waves and, therefore, must be sensitive to the numerical boundary conditions and the precise form of the spatial discretization. The instability is predicted by linear Fourier analysis, which is not completely appropriate for this relaxation scheme. However, numerical experiments on the nonlinear equations confirm the long-wave instability.

The instability can be suppressed by a multigrid scheme. However, in that case damped Symmetric Point Gauss–Seidel is a better choice, because it is stable as a single-grid scheme and can produce similar multigrid convergence factors at a much lower cost [9]. This scheme, however, does not provide good convergence in cases of strong alignment, the flow being aligned with the grid.

A uniformly good convergence rate can be obtained with damped Alternating Direction Line Jacobi, which has about the same cost as SLGS. The two-level analysis indicates a convergence factor that is at worst 0.562.

The use of SLGS has been recommended for the Navier-Stokes equations in [5]. Numerical experiments in [6] suggest grid-independent convergence rates. Because the

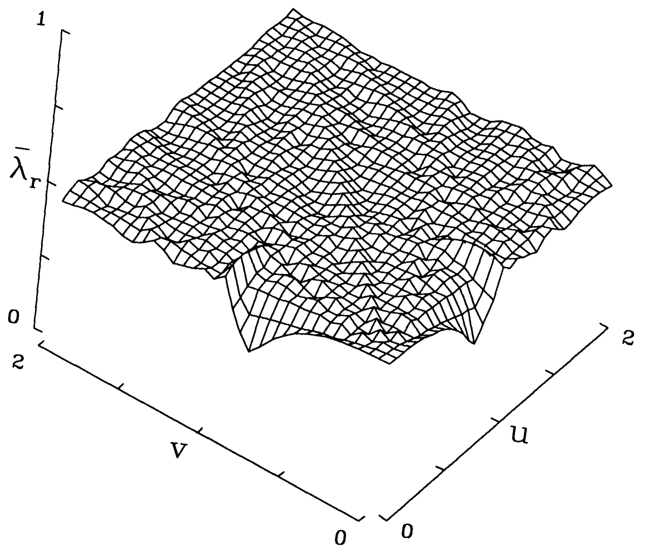


FIG. 4. *Multigrid convergence factor for* ADLJ *relaxation as a function of u and v* ($c = 1$). *Each point is computed for* 64×64 *grid.*

spatial discretization is essentially different in the long-wave regime, due to viscous terms and a different form of flux-splitting, the present analysis does not automatically carry over to this situation.

Finally, it should be noted that the results of two-level analysis tend to be too pessimistic. If one considers a wave perpendicular to a streamline, i.e., $u\theta_x = -v\theta_y$, and concentrates on the long waves (small θ_x and θ_y), the coarse-grid correction operator $\hat{K}$ can be approximated by $\hat{K} \simeq I - (\hat{L}^{2h})^\dagger \hat{L}^h$, which has an eigenvalue $\frac{1}{2}$. This value will dominate the results of two-level analysis, as can be seen in Fig. 4. However, the eigenvalue corresponds to a wave for which the exact operator vanishes and, therefore, describes convergence of the truncation error. If one is only interested in convergence to a level where the iteration error is of the order of the truncation error, this value is clearly not important. The reader is referred to [10], [11], [12] for further details and applications.

REFERENCES

[1] R. W. BEAM AND R. F. WARMING, *An implicit finite-difference algorithm for hyperbolic systems in conservation-law form*, J. Comput. Phys., 22 (1976), pp. 87-110.
[2] A. BRANDT, *Guide to multigrid development*, Lecture Notes in Math., 960 (1981), pp. 220-312.
[3] S. R. CHAKRAVARTHY, *Relaxation methods for unfactored implicit upwind schemes*, AIAA paper no. 84-0165 (1984).
[4] J. DOUGLAS AND J. GUNN, *A general formulation of alternating direction methods*, I, Numer. Math., 6 (1964), pp. 87-110.

[5] R. W. MACCORMACK, *Current status of numerical solutions of the Navier-Stokes equations*, AIAA paper no. 85-0032 (1985).

[6] Y. J. MOON, *Grid size dependence on convergence for computation of the Navier-Stokes equations*, AIAA J., 24 (1986), pp. 1705-1706.

[7] W. A. MULDER, *Multigrid relaxation for the Euler equations*, J. Comput. Phys., 60 (1985), pp. 235-252.

[8] ———, *Computation of the quasi-steady gas flow in a spiral galaxy by means of a multigrid method*, Astron. and Astrophys., 156 (1986), pp. 354-380.

[9] ———, *Analysis of a multigrid method for the Euler equations of gas dynamics in two dimensions*, in Multigrid Methods: Theory, Applications, and Supercomputing, S. McCormick, ed., Marcel Dekker, New York, 1988, pp. 467-489.

[10] ———, *A new multigrid approach to convection problems*, J. Comput. Phys., 83 (1989), pp. 303–323.

[11] ———, *Multigrid, alignment, and Euler's equations*, presented at the Fourth Copper Mountain Conference on Multigrid Methods, Copper Mountain, Colorado, April 1989. To be published by SIAM.

[12] ———, *A high resolution Euler solver*, AIAA paper no. 89-1949 (1989).

[13] J. L. THOMAS, B. VAN LEER, AND R. W. WALTERS, *Implicit flux-split schemes for the Euler equations*, AIAA paper no. 85-1680 (1985).

[14] B. VAN LEER, *Flux-vector splitting for the Euler equations*, Lecture Notes in Physics, 170 (1982), pp. 507-512.

[15] B. VAN LEER AND W. A. MULDER, *Relaxation methods for hyperbolic conservation laws*, in Numerical Methods for the Euler Equations of Fluid Dynamics, F. Angrand, A. Dervieux, J. A. Desideri, and R. Glowinski, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1985, pp. 312-333.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 3, pp. 399-424, May 1990

VORTEX METHODS FOR SLIGHTLY VISCOUS THREE-DIMENSIONAL FLOW*

DALIA FISHELOV†

Abstract. Vortex methods for slightly viscous three-dimensional flow are presented. Vortex methods have been used extensively for two-dimensional problems, though their most efficient extension to three-dimensional problems is still under investigation. A method that evaluates the vorticity by exactly differentiating an approximate velocity field is applied. Numerical results are presented for a flow past a semi-infinite plate, and they demonstrate three-dimensional features of the flow and transition to turbulence.

Key words. vortex methods, boundary layers, turbulent flow

AMS(MOS) subject classifications. 76D05, 76D10, 35Q10

1. Introduction. Vortex methods as suggested by Chorin [12] were applied to various problems to simulate incompressible flows (see [34] and [32] for a review). These grid-free methods represent complicated flows by concentrating the computational elements in regions where small-scale phenomena predominate and few elements elsewhere. In addition, vortex methods introduce no artificial viscosity, and therefore they are adequate for solving the slightly viscous Navier–Stokes equation.

Vortex methods have been used extensively in the last 15 years, especially for two-dimensional flows. Although three-dimensional vortex methods have been considered inherently difficult, we represent a scheme that involves no elaborate computations and is a natural extension of the two-dimensional schemes. We applied this method to a three-dimensional flow past a semi-infinite plate at high Reynolds number. The velocity far away from the plate is assumed to be uniform. If we assume that the flow is independent of the spanwise variable, the problem is two-dimensional, otherwise the flow is three-dimensional. Chorin [11]-[13] solved the two-dimensional problem numerically; he used computational elements, called blobs, with a smoothed kernel. This kernel is obtained by convolving the singular kernel, which connects vorticity and velocity, with a smoothing function (called a cutoff function). The latter approximates a delta function in the sense that a finite number of its moments are identical to those of a delta function.

A numerical solution to a three-dimensional problem was introduced in 1980 by Chorin [11] and by Leonard [32]-[34] using different vortex filament methods. In the filament method we approximate the initial velocity and vorticity along vortex lines, whose tangents are parallel to the vorticity vector. Since circulation is conserved along vortex lines, there is no need to update vorticity. Both authors [11], [34] stepped the Navier–Stokes equations in time by splitting them to the Euler and the heat equations. In [33] Leonard introduces one of the earliest vortex methods to solve the inviscid three-dimensional Euler equations numerically. In his computations he was able to simulate the time development of spotlike disturbances in laminar three-dimensional boundary layer. He suggested to split the velocity field into a sum of the velocity at infinity and a perturbed one, and to track vortex lines and compute their curvatures.

* Received by the editors April 25, 1988; accepted for publication (in revised form) May 2, 1989. This work was partially supported (at the Lawrence Berkeley Laboratory) by the Applied Mathematical Sciences Subprogram of the Office of Energy Research, U.S. Department of Energy under contract DE-AC03-76SF00098, and in part by the Office of Naval Research under contract N00014-76-C-0316.

† Department of Mathematics and Lawrence Berkeley Laboratory, University of California, Berkeley, California 94720. Present address, Department of Applied Mathematics, The Weizmann Institute of Science, P.O. Box 26, Rehovot 76100, Israel.

He extended his method to the viscous case [34] using a core spreading technique, in which the core of the filaments was changed every timestep to satisfy the heat equation. This scheme was proved to approximate the wrong equations, rather than the Navier-Stokes equations [22].

Chorin suggested a different filament method to solve the three-dimensional problem. He approximates vortex lines by segments and then, using the Biot-Savart law, he updates the endpoints of the segments for the Euler equation at every timestep. The heat equation is approximated in the statistical sense via a random-walk algorithm. Since Chorin uses segments to approximate vortex lines, his algorithm involves no elaborate calculations, such as evaluation of curvatures. However it is not highly accurate in space. The purpose of this paper is to modify Chorin's scheme to gain higher spatial accuracy.

Following Beale and Majda [4], [5] and Anderson and Greengard [1], [2], we achieve higher spatial accuracy by generalizing the two-dimensional blobs to three-dimensional ones. Vorticity as well as blob locations must be updated at every timestep. Two versions of the three-dimensional blob extension were suggested. Beale and Majda suggested approximating spatial derivatives with finite differences, whereas Anderson explicitly differentiates the smoothed kernel mentioned above. We chose to apply the method of Anderson, since it eliminates one source of error, associated with spatial differentiation. The algorithm and its accuracy is then similar to the two-dimensional one. The results shown here are the first attempt to apply this scheme numerically. Convergence was proved in [3] and [10] for the Euler equations. Applying the convergence proofs to our scheme, we show that for smooth cutoff functions second-order accuracy in space is gained. Higher-order space accuracy can be achieved by using cutoff functions, in which more moments agree with those of a delta function. We were able to resolve three-dimensional features of the flow and transition to turbulence. The numerical results are in agreement with experimental results shown in [25], which suggest that at high Reynolds numbers there exist a large number of small hairpins.

Spectral methods, which are highly accurate for smooth flows, were used for turbulent flows by Orszag and Kells [35] and Orszag and Patera [36]. In [35] and [36] periodic boundary conditions in the streamwise direction were assumed. Note that nonperiodic boundary conditions in the streamwise direction might impose nonsmoothness of the solution, such as that of the Blasius solution at the leading edge. This nonsmoothness must be carefully treaded when using a spectral method. If we use a finite-difference scheme, it requires a mesh that is inversely proportional to the square root of the Reynolds number. However, a new finite-difference scheme with local mesh refinement has recently been developed by Bell, Colella, and Glaz [7] and its application for three-dimensional flows with transition to turbulence need to be tested.

The paper is organized as follows. In § 2 we represent the fundamental equations, in § 3 the numerical scheme, and in § 4 we describe the boundary conditions. In § 5 we show that if we use a smooth cutoff function, second-order space accuracy is assured for the Euler equations. The error from the viscous term is discussed as well. We also suggest a new way for treating this term. Section 6 represents numerical results and § 7 concludes the paper.

2. Representation of the problem. The flow is described by the Navier-Stokes equations, formulated for the vorticity ξ:

$$\begin{aligned} &\partial_t \xi + (\mathbf{u} \cdot \nabla)\xi - (\xi \cdot \nabla)\mathbf{u} = R^{-1}\Delta \xi, \\ &\operatorname{div} \mathbf{u} = 0, \end{aligned} \tag{2.1}$$

where $\xi = \operatorname{curl} \mathbf{u}$, $\mathbf{u} = (u, v, w)$ is the velocity vector, $\mathbf{r} = (x, y, z)$ is the position vector and $\Delta = \nabla^2$ is the Laplace operator. $R = UL/\nu$ is the Reynolds number, where U and L are typical velocity and length, respectively, and ν is the viscosity.

We will solve the above equations for a flow past a semi-infinite flat plate located at $z = 0$, $x \geqq 0$. Far away from the plate (for $z \to \infty$) there is a uniform flow in the positive x direction, i.e.,

$$\mathbf{u} = (U_\infty, 0, 0) \quad \text{for } z \to \infty, \quad t > 0.$$

On the plate we impose the no-leak boundary condition $\mathbf{u} \cdot \mathbf{n} = 0$, where $\mathbf{n}$ is normal to the plate. We also impose the no-slip boundary condition $\mathbf{u} \cdot \mathbf{s} = 0$, where $\mathbf{s}$ is tangential to the plate. Initially, $\mathbf{u} = (U_\infty, 0, 0)$ at $t = 0$.

The Prandtl equations are known to approximate the Navier-Stokes equations near the plate, and are used therefore in a thin layer $0 \leqq z \leqq z_0$. The Navier-Stokes equations are employed in the region $z \geqq z_0$. In the Prandtl equations we assume that $\xi = (\xi_1, \xi_2, 0)$, i.e., ξ_3 is negligible in comparison to the other components (see, e.g., [39]). Thus

$$\begin{aligned} &\partial_t \xi_1 + (\mathbf{u} \cdot \nabla)\xi_1 = R^{-1} \partial^2_{zz} \xi_1, \\ (2.2) \qquad &\partial_t \xi_2 + (\mathbf{u} \cdot \nabla)\xi_2 = R^{-1} \partial^2_{zz} \xi_2, \\ &\operatorname{div} \mathbf{u} = 0, \end{aligned}$$

$$(2.3) \qquad \xi_1 = -\frac{\partial v}{\partial z}, \quad \xi_2 = \frac{\partial u}{\partial z}, \quad \mathbf{u} = (u, v, w).$$

The Pradtl equations admit the two-dimensional steady state solution—the Blasius solution. However, the three-dimensional Navier-Stokes equations are unstable at high Reynolds numbers ($R \geqq 1000$), i.e., small perturbations in the Blasius solution may cause large perturbations in the solution as time progresses. Once the disturbances in the Blasius solution begin to grow, spanwise vortices appear, the solution then depends on the spanwise variable y, and there is a transition to turbulence.

Theoretical aspects of this instability are given in Benney and Lin [9] and Benney [8]; they suggest that the secondary motions produced by the interaction of three-dimensional modes with two-dimensional ones can produce profiles that are highly unstable. Physical experiments done by Kline et al. [31], Klebanoff, Tidstrom, and Sargent [30], and Head and Bandyapodhyay [25] showed that secondary motion, caused by the production of longitudinal vorticity due to three-dimensional disturbances, creates highly unstable profiles leading to turbulent spots. Klebanoff, Tidstrom, and Sargent [30] suggested that the weak three-dimensional disturbances may control the nonlinear development of the flow and its transition to turbulence.

Kinney and Paolino [28], Schmall and Kinney [43], and Kinney and Cielak [29] suggested vorticity formulation along with a boundary condition for the vorticity on the body. Recently, van der Vegt [44] performed two- and three-dimensional calculations using a vortex-spectral method. He simulated the flow over a cylinder and pointed out the ability of the vortex model to describe typical viscous phenomena, such as flow separation.

As the outcome of numerical and physical experiments two main models of the turbulent boundary layer have emerged. One is the coherent structure with large horseshoes [27], and the other is the uncorrelated hairpins [14], [42]. Using a numerical simulation, Kim and Moin [27] found out that the bursting process is associated with well-organized vortical structures described by large horseshoes. On the other hand, Chorin [14], Siggia [42], Kerr [26], and Head and Bandyapodhyay [25] claim that the structure is better described by uncorrelated small hairpins.

The experiments of Head and Bandyapodhyay [25] for high Renyolds numbers ($R \geqq 1{,}000$) indicate the existence of large numbers of vortex pairs or hairpin vortices, extending through at least a substantial part of the boundary-layer thickness; for the most part they are inclined to the wall at a characteristic angle of 40° to 50°. At low Reynolds number ($R \leqq 800$) the hairpins are much less elongated and are better described as horseshoe vortices or vortex loops. Head and Bandyapodhyay [25] note that almost all investigators have used experimental techniques that limit the observations to relatively low Reynolds numbers, where the structure is markedly different from that at high Reynolds numbers; vortex lines tend to appear as low aspect-ratio loops rather than extended vortex pairs or hairpins. In our calculations, we found support to the hairpins model, rather than to the horseshoe model.

One of the conclusions from the experimental data in [31] is that the flow is periodic in the spanwise direction. We therefore solve (2.1) and (2.2) with the following periodic boundary condition:

$$\mathbf{u}(x, y+q, z) = \mathbf{u}(x, y, z), \qquad \xi(x, y+q, z) = \xi(x, y, z).$$

As was noted in [11], q was found to be roughly 0.1.

3. The numerical scheme. We first describe the random-vortex method for the Navier-Stokes equations and then the three-dimensional sheet method, called the tile method, for the Prandtl equations.

3.1. Time discretization. We split the Navier-Stokes equations into the Euler equations and the heat equation. The Euler equation (3.1) governs the flow of an inviscid fluid:

$$\partial_t \xi + (\mathbf{u} \cdot \nabla)\xi - (\xi \cdot \nabla)\mathbf{u} = 0. \tag{3.1}$$

Note that for a two-dimensional case the last term in the left-hand side of (3.1) vanishes, and therefore vorticity is a material property, i.e., $D\xi/Dt = \partial\xi/\partial t + (\mathbf{u} \cdot \nabla)\xi = 0$. However, this is not necessarily true in three dimensions.

The heat equation is

$$\frac{\partial \xi}{\partial t} = R^{-1}\,\Delta\xi \tag{3.2}$$

(it is also called the diffusion equation). Both (3.1) and (3.2) are easier to analyze than the Navier-Stokes equations. We apply a Strang-type scheme to step the Navier-Stokes equations in time, using (3.1) and (3.2). This is done in the following way: we represent both problems above in the form

$$\xi_t = A(\xi).$$

For the first one

$$A(\xi) = A_1(\xi) = (\xi \cdot \nabla)\mathbf{u},$$

and for the second

$$A(\xi) = A_2(\xi) = R^{-1}\,\Delta\xi.$$

For both operators we apply the modified Euler scheme:

$$\xi^{n+1/2} = \xi^n + \frac{\Delta t}{2} A(\xi^n), \qquad \xi^{n+1} = \xi^n + \Delta t A(\xi^{n+1/2}).$$

Let $L(\Delta t)$ be the operator that acts on ξ^n to yield ξ^{n+1}, i.e.,

$$L(\Delta t)\xi^n = \xi^{n+1} = \xi^n + \Delta t A\left(\xi^n + \frac{\Delta t}{2} A(\xi^n)\right).$$

$L_1(\Delta t)$, $L_2(\Delta t)$ are defined as $L(\Delta t)$ with A_1, A_2 replacing A. We finally arrive at the following scheme for discretizating (2.1) in time:

$$\xi^{n+1} = L_1\left(\frac{\Delta t}{2}\right) L_2\left(\frac{\Delta t}{2}\right) L_2\left(\frac{\Delta t}{2}\right) L_1\left(\frac{\Delta t}{2}\right) \xi^n.$$

According to [21], this scheme is second-order accurate in time, is accurate up to order two in the time variable, even in the nonlinear case. The same time discretization was used also in [17] and [18].

3.2. Spatial discretization.

3.2.1. The Euler equations. For an incompressible fluid the following relation ((3.7) below) between vorticity and velocity holds [15]. Since $\operatorname{div} \mathbf{u} = 0$, there exists a function ψ, called a stream function, such that

$$\mathbf{u} = \nabla \times \psi, \tag{3.3}$$

and we may choose ψ such that $\operatorname{div} \psi = 0$. By definition

$$\xi = \nabla \times \mathbf{u}, \tag{3.4}$$

and therefore, from (3.3), we find that

$$\Delta \psi = -\xi. \tag{3.5}$$

Thus we may determine the velocity from the vorticity by first solving the Poisson equation (3.5), and then applying (3.3).

If G is a fundamental solution of the Laplace equation, then

$$\psi = G * \xi = \int G(\mathbf{x}-\mathbf{x}')\xi(\mathbf{x}')\, d\mathbf{x}', \tag{3.6}$$

where $G(\mathbf{x}) = -1/4\pi|\mathbf{x}|$, $\mathbf{x} = (x, y, z)$, and the integration is taken over the whole three-dimensional space. Substituting (3.6) in (3.3), we find

$$\mathbf{u}(\mathbf{x}, t) = \int K(\mathbf{x}-\mathbf{x}')\xi(\mathbf{x}', t)\, d\mathbf{x}', \tag{3.7}$$

where

$$K(\mathbf{x}) = -\frac{1}{4\pi|\mathbf{x}|^3}\begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix}. \tag{3.8}$$

Note that (3.7) is a consequence of incompressibility only.

In vortex methods particle trajectories are followed. Let $\mathbf{x}(\boldsymbol{\alpha}, t)$ be the trajectory of a particle in the fluid that is at the point $\boldsymbol{\alpha}$ at $t = 0$. For fixed $\boldsymbol{\alpha}$ the trajectory $\mathbf{x}(\boldsymbol{\alpha}, t)$ is obtained from the velocity field $\mathbf{u}$ as a solution of the ordinary differential equation:

$$\frac{d\mathbf{x}}{dt}(\boldsymbol{\alpha}, t) = \mathbf{u}(\mathbf{x}(\boldsymbol{\alpha}, t), t), \qquad \mathbf{x}(\boldsymbol{\alpha}, 0) = \boldsymbol{\alpha}. \tag{3.9}$$

Combining (3.7) and (3.8), we find

$$\begin{aligned}\frac{d\mathbf{x}}{dt} &= \int K(\mathbf{x}(\boldsymbol{\alpha}, t) - \mathbf{x}'(\boldsymbol{\alpha}, t))\xi(\mathbf{x}', t)\, d\mathbf{x}' \\ &= \int K(\mathbf{x}(\boldsymbol{\alpha}, t) - \mathbf{x}(\boldsymbol{\alpha}', t))\xi(\mathbf{x}(\boldsymbol{\alpha}', t), t)\, d\boldsymbol{\alpha}'.\end{aligned} \tag{3.10}$$

The last equality is true, since for an incompressible fluid the Jacobian of the transformation $\boldsymbol{\alpha}(t) \to \mathbf{x}(\boldsymbol{\alpha}, t)$ is the identity.

We must supply initial conditions to (3.10). We therefore set the initial velocity and vorticity on a regular mesh:

$$\alpha_i = (h_1 i_1, h_2 i_2, h_3 i_3), \qquad i = 1, \cdots, n,$$
$$1 \leqq i_1 \leqq N_1, \quad 1 \leqq i_2 \leqq N_2, \quad 1 \leqq i_3 \leqq N_3,$$

and then track these particles in Lagrangian coordinates. To discretize the equations, we set $\xi = \sum_j \xi_j$, where the ξ_j are functions of small support. Let κ_j be the intensity of the jth particle, i.e., $\kappa_j = \int \xi_j \, dx\, dy\, dz$. Then we obtain the following set of ordinary differential equations for the approximate locations of the particles $\tilde{\mathbf{x}}_i$:

$$\begin{aligned}\frac{d\tilde{\mathbf{x}}_i}{dt}(t) &= \tilde{\mathbf{u}}_i(t) = \sum_{j=1}^{n} K_\delta(\tilde{\mathbf{x}}_i(t) - \tilde{\mathbf{x}}_j(t))\tilde{\kappa}_j(t), \\ \tilde{\mathbf{x}}_i(0) &= \boldsymbol{\alpha}_i,\end{aligned} \tag{3.11}$$

where $\phi: R^3 \to R$, $\phi_\delta = (1/\delta^3)\phi(\mathbf{x}/\delta)$ is the cutoff function, and $K_\delta = K * \phi_\delta$ is a smoothed kernel. K_δ replaces the kernel K (defined in (3.8)), which is singular at $\mathbf{x} = 0$. Here $\tilde{\kappa}_j(t)$, $\tilde{\mathbf{x}}_j(t)$ approximate $\kappa_j(t)$ and $\mathbf{x}_i(t)$, respectively, the exact intensity and particle locations for the Euler equations.

We may write K_δ in the following way:

$$K_\delta(\mathbf{x}) = K(\mathbf{x}) f_\delta(\mathbf{x}), \tag{3.12}$$

where $f_\delta(\mathbf{x}) = (1/\delta^3) f(\mathbf{x}/\delta)$. If $f(\mathbf{x})$ is chosen to be radially symmetric, the relation between ϕ and f is $\phi(r) = f'(r)/4\pi r^2$ (see [6]). We specify $f(\mathbf{x}) = f(r)$ below:

$$f(r) = \begin{cases} 1, & r \geqq 1, \\ \frac{5}{2} r^3 - \frac{3}{2} r^5, & r < 1. \end{cases} \tag{3.13}$$

This function is continuous with its first derivative at $r = 1$. Substituting (3.8) and (3.13) in (3.12) yields

$$K_\delta = -\frac{1}{4\pi |\mathbf{x}|^3} \begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix} \quad \text{for } |\mathbf{x}| > \delta,$$

and

$$K_\delta = -\frac{1}{4\pi |\mathbf{x}|^3 \delta^3} \begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix} \cdot \left(\frac{5}{2}\left(\frac{r}{\delta}\right)^3 - \frac{3}{2}\left(\frac{r}{\delta}\right)^5 \right) \quad \text{for } |\mathbf{x}| < \delta. \tag{3.14}$$

For a three-dimensional Euler equation vorticity is not a material quantity, and therefore we must track vorticity as well as blob locations. We use the equation

$$\frac{d\xi}{dt} = (\xi \cdot \nabla)\mathbf{u}.$$

Therefore, the evolution of vorticity along particle trajectories is described by the equations

$$\frac{d\xi}{dt}(\mathbf{x}(\boldsymbol{\alpha}, t), t) = (\xi(\mathbf{x}(\boldsymbol{\alpha}, t), t) \cdot \nabla_{\mathbf{x}})\mathbf{u}(\mathbf{x}(\boldsymbol{\alpha}, t), t), \tag{3.15}$$

where $\nabla_{\mathbf{x}}$ is the gradient with respect to the Eulerian coordinates. Applying (3.11), we find that the following equality holds for the approximated velocity $\tilde{\mathbf{u}}$

$$\nabla_{\mathbf{x}}\tilde{\mathbf{u}}(\mathbf{x}, t) = \sum_{j=1}^{n} \nabla_{\mathbf{x}} K_\delta(\mathbf{x} - \tilde{\mathbf{x}}_j(t))\tilde{\kappa}_j(t),$$

where $\nabla_{\mathbf{x}} K_\delta$ is derived analytically in Eulerian coordinates using the definition of K_δ (3.15). Substitution of the last equality in (3.14) yields

$$\frac{d\tilde{\kappa}_i}{dt} = \sum_{j=1}^{n} (\tilde{\kappa}_i \cdot \nabla_{\mathbf{x}})K_\delta(\tilde{\mathbf{x}}_i(t) - \tilde{\mathbf{x}}_j(t))\tilde{\kappa}_j(t). \tag{3.16}$$

This can be written in the form

$$\frac{d\tilde{\kappa}_i}{dt} = \sum_{j=1}^{n} (\tilde{\kappa}_i^x A(\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j)\tilde{\kappa}_j(t) + \tilde{\kappa}_i^y B(\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j)\tilde{\kappa}_j(t) + \tilde{\kappa}_i^z C(\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j)\tilde{\kappa}_j(t)), \tag{3.17}$$

where $\tilde{\kappa}_i = (\tilde{\kappa}_i^x, \tilde{\kappa}_i^y, \tilde{\kappa}_i^z)$, and

$$A(\mathbf{x}) = \frac{\partial}{\partial x} K_\delta(\mathbf{x}), \quad B(\mathbf{x}) = \frac{\partial}{\partial y} K_\delta(\mathbf{x}), \quad C(\mathbf{x}) = \frac{\partial}{\partial z} K_\delta(\mathbf{x}).$$

Or more explicitly,

$$\begin{aligned}
\tilde{\kappa}_i^x A(\mathbf{x})\tilde{\kappa}_j &= \frac{\tilde{\kappa}_i^x}{4\pi|\mathbf{x}|^5}(-(|\mathbf{x}|^2 - 3x^2), 3xy, 3xz) \times \tilde{\kappa}_j, \\
\tilde{\kappa}_i^y B(\mathbf{x})\tilde{\kappa}_j &= \frac{\tilde{\kappa}_i^y}{4\pi|\mathbf{x}|^5}(3yx, -(|\mathbf{x}|^2 - 3y^2), 3yz) \times \tilde{\kappa}_j, \\
\tilde{\kappa}_i^z C(\mathbf{x})\tilde{\kappa}_j &= \frac{\tilde{\kappa}_i^z}{4\pi|\mathbf{x}|^5}(3zx, 3zy, -(|\mathbf{x}|^2 - 3z^2)) \times \tilde{\kappa}_j,
\end{aligned} \tag{3.18}$$

for $|\mathbf{x}| < \delta$, and

$$\begin{aligned}
\tilde{\kappa}_i^x A(\mathbf{x})\tilde{\kappa}_j &= \frac{\tilde{\kappa}_i^x}{4\pi\delta^5}(-(2.5\delta^2 - 1.5|\mathbf{x}|^2 - 3x^2), 3xy, 3xz) \times \tilde{\kappa}_j, \\
\tilde{\kappa}_i^y B(\mathbf{x})\tilde{\kappa}_j &= \frac{\tilde{\kappa}_i^y}{4\pi\delta^5}(3yx, -(2.5\delta^2 - 1.5|\mathbf{x}|^2 - 3y^2), 3yz) \times \tilde{\kappa}_j, \\
\tilde{\kappa}_i^z C(\mathbf{x})\tilde{\kappa}_j &= \frac{\tilde{\kappa}_i^z}{4\pi\delta^5}(3zx, 3zy, -(2.5\delta^2 - 1.5|\mathbf{x}|^2 - 3z^2)) \times \tilde{\kappa}_j,
\end{aligned} \tag{3.19}$$

for $|\mathbf{x}| > \delta$. To conclude, the semidiscrete three-dimensional scheme that we used for the Euler equation is

$$\begin{aligned} &\frac{d\tilde{\mathbf{x}}_i}{dt}(t) = \tilde{\mathbf{u}}_i(t) = \sum_{i=1}^{n} K_\delta(\tilde{\mathbf{x}}_i(t) - \tilde{\mathbf{x}}_j(t))\tilde{\kappa}_j(t), \\ &\frac{d\tilde{\kappa}_i}{dt} = \sum_{j=1}^{n} (\tilde{\kappa}_i \cdot \nabla_{\mathbf{x}}) K_\delta(\tilde{\mathbf{x}}_i(t) - \tilde{\mathbf{x}}_j(t))\tilde{\kappa}_j(t), \\ &\tilde{\mathbf{x}}_i(0) = \boldsymbol{\alpha}_i, \qquad \tilde{\kappa}_i(0) = \kappa_i^0, \end{aligned} \tag{3.20}$$

where K_δ is defined in (3.14), and the second equation is given in more detail in (3.17)-(3.19). Here, κ_i^0 are initial values of the intensities of the computational elements on the initial grid.

3.2.2. The heat equation. The second equation to solve is the heat equation:

$$\frac{\partial \xi}{\partial t} = R^{-1}\Delta\xi \quad \text{or} \quad \frac{\partial \mathbf{u}}{\partial t} = R^{-1}\Delta\mathbf{u}.$$

Following Chorin ([11] and [13]) we use the random-walk method to step the heat equation in time, i.e., we move the blobs according to

$$\tilde{\mathbf{x}}_i^{n+1} = \tilde{\mathbf{x}}_i^n + \boldsymbol{\eta}(\Delta t),$$

where $\boldsymbol{\eta}(\Delta t) = (\eta_1(\Delta t), \eta_2(\Delta t), \eta_3(\Delta t))$ and η_1, η_2, η_3 are Gaussian random variables with mean zero and variance $2\Delta t/R$, chosen independently of each other.

Note that we use the trapezoidal rule in (3.11) and (3.16) to approximate spatial integrals. The error due to this approximation depends on the derivatives of the integrands, and in particular on the voriticity, i.e., if the vorticity grows so does the error. Therefore, if we find that the vorticity grows while using blobs for the Navier-Stokes equations, we replace a blob that carries a high enough vorticity with several blobs. The new blobs are placed at the same computational point, and share the same total vorticity of the original blob. Since the random walk is used to simulate the heat equation, these blobs will likely find themselves in different locations at the next timestep. If we use filaments, growth in vorticity causes stretching of the filaments. In this case we should split the vortex line into several short ones, and then use some interpolation between the endpoints of the old filament to keep a desired accuracy. This interpolation is an additional source of error, but it can be avoided if we adopt the three-dimensional vortex blob method described above.

3.3. Prandtl equations. The Prandtl equations (2.2) used in a thin layer $0 \leqq z \leqq z_0$ above the plate were solved numerically by the tile method, which is the three-dimensional extension of the sheet method (see [11], [13]). This was done to evaluate the boundary conditions on the plate, since it was found in [13] and [11] that blobs did not accurately represent the velocity field near the boundary. We describe the tile method for a region $0 \leqq z \leqq \infty$, noting that the boundary conditions at $z = z_0$ will be viewed as those at infinity, seen from the plate.

In the tile method the computational elements are rectangles, parallel to the plate, that represent a jump in the velocity components u, v. Thus (ξ_1, ξ_2) is the intensity of the tile, where $\xi_2 = u_{\text{above}} - u_{\text{below}}$, $\xi_1 = v_{\text{above}} - v_{\text{below}}$. Consider a collection of N tiles T_i, with intensities $((\xi_1)_i, (\xi_2)_i)$, $i = 1, \cdots, N$ and centers $\mathbf{x}_i = (x_i, y_i, z_i)$. The motions of these tiles are described by (2.3), i.e.,

$$\xi_1 = -\frac{\partial v}{\partial z}, \qquad \xi_2 = \frac{\partial u}{\partial z},$$

and if we integrate these equations with respect to z, we have

$$u(x, y, z, t) = u_\infty(x, y, t) - \int_z^\infty \xi_2(x, y, z')\, dz', \tag{3.21}$$

$$v(x, y, z, t) = v_\infty(x, y, t) + \int_z^\infty \xi_1(x, y, z')\, dz', \tag{3.22}$$

where $u_\infty(x, y, t)$, $v_\infty(x, y, t)$ are the velocity components u, v as $z \to \infty$. By incompressibility and the boundary condition $w(x, y, 0, t) = 0$, we have

$$w(x, y, z, t) = -\partial_x \int_0^z u(x, y, z')\, dz' - \partial_y \int_0^z v(x, y, dz')\, dz'. \tag{3.23}$$

Equations (3.21)-(3.23) provides a relation between the vorticity and the velocity, which replaces the one given by (3.7) for the interior region.

The above equations can be approximated by

$$u_i = \tilde{u}(x_i, y_i, z_i, t) = u_\infty(x_i, y_i, t) - \frac{1}{2}(\xi_2)_i - \sum_j (\xi_2)_j d_j f_j, \tag{3.24}$$

$$v_i = \tilde{v}(x_i, y_i, z_i, t) = v_\infty(x_i, y_i, t) + \frac{1}{2}(\xi_1)_i + \sum_j (\xi_1)_j d_j f_j, \tag{3.25}$$

where $d_j = 1 - |x_i - x_j|/h_1$, and $f_j = 1 - |y_i - y_j|/h_2$ are smoothing functions, the summations in (2.24)-(2.25) are over all T_j for which $0 \leqq d_j \leqq 1$, $0 \leqq f_j \leqq 1$, and $z_j \geqq z_i$.

Similarly, from (3.23)

$$w_i = \tilde{w}(x_i, y_i, z_i, t) = -(I_+ - I_-)/h_1 - (J_+ - J_-)/h_2,$$

where

$$I_\pm = u_\infty(x_i \pm h_1/2, y_i, t) z_i - \textstyle\sum_\pm^x (\xi_2)_j d_j^\pm f_j z_j^*,$$

$$J_\pm = v_\infty(x_i, y_i \pm h_2/2, t) z_i + \textstyle\sum_\pm^y (\xi_1)_j d_j f_j^\pm z_j^*,$$

and

$$d_j^\pm = 1 - \frac{|x_i \pm h_1/2 - x_j|}{h_1}, \quad f_j^\pm = 1 - \frac{|y_i \pm h_2/2 - y_j|}{h_2}, \quad z_j^* = \min(z_i, z_j).$$

The sums $\sum_+^x, \sum_-^x$ are over all T_i, such that $0 \leqq f_j^\pm \leqq 1$, and $0 \leqq d_j^+ \leqq 1$, $0 \leqq d_j^- \leqq 1$, respectively. Similarly, the sums $\sum_+^y, \sum_-^y$ are over all T_i, such that $0 \leqq d_j^\pm \leqq 1$, and $0 \leqq f_j^+ \leqq 1$, $0 \leqq f_j^- \leqq 1$, respectively. This is a thin vertical layer, and therefore the number of operations to calculate the velocity fields for the tile method is $O(N)$.

For simplicity, we describe the motion of a tile for a first-order timestepping Euler scheme

$$x_i^{n+1} = x_i^n + \Delta t \cdot u_i, \qquad y_i^{n+1} = y_i^n + \Delta t \cdot v_i,$$

$$z_i^{n+1} = z_i^n + \Delta t \cdot w_i + \eta(\Delta t),$$

where η is a Gaussian random variable with mean zero and variance $2\Delta t/R$. Note that η appears only in the z component, since the Prandtl equation (2.2) assumes that vorticity diffuses in the z direction only.

4. Boundary conditions. We first specify the boundary conditions for the region $z \geqq z_0$, in which the Navier-Stokes equations are used. At infinity the flow is uniform and is in the x direction, i.e., $\mathbf{u}(x, y, z, t) \to (U_\infty, 0, 0)$ as $z \to \infty$. Boundary conditions also have to be imposed at $z = z_0$ (see [39, p. 111]), and they link the two computational regions. If a tile finds itself in the region $z \geqq z_0$ after taking a timestep, it turns into a blob. Similarly, if a blob enters the thin layer in which the Prandtl equations are employed, it becomes a tile. We assign the same circulation to a tile which turns into a blob and vice versa. Thus $\kappa_i = \xi_i h_1 h_2$, where κ_i is the intensity of the blob, and ξ_i is the intensity of the tile. In addition, we require continuity of u and v at $z = z_0$.

The boundary conditions for the Prandtl equations are

(a) $u(x, y, z_0, t) = u_\infty(x, y, t)$, and $v(x, y, z_0, t) = v_\infty(x, y, t)$, where $u_\infty(x, y, t)$ and $v_\infty(x, y, t)$ are calculated by the blobs, located at $z \geqq z_0$.

(b) $\mathbf{u} \cdot \mathbf{n} = 0$ at $z = 0$, where $\mathbf{n}$ is normal to the plate. This is done by the method of images, i.e., for each blob or tile at (x, y, z), carrying vorticity $\xi(x, y, z)$ we add an imaginary blob or tile at (x, y, z) with vorticity $-\xi(x, y, z)$.

(c) $\mathbf{u} \cdot \mathbf{s} = 0$ at $z = 0$, where $\mathbf{s}$ is tangential to the plate $z = 0$. This is done by creating tiles at the boundary, assigning vorticity to each of them (see [11]). In more detail: we calculate $u_0 = u(x, y, 0) = u_\infty(x, y, t) - \int_0^\infty \xi_2 \, dz$ and $v_0 = v(x, y, 0) = v_\infty(x, y, t) + \int_0^\infty \xi_1 \, dz$, and replace the integrals $\int_0^\infty \xi_2 \, dz$ and $\int_0^\infty \xi_1 \, dz$ by the sums $\sum_{j=1}^\infty (\xi_2)_j d_j f_j$ and $\sum_{j=1}^\infty (\xi_1)_j d_j f_j$, respectively. The only tiles that contribute to these sums are those located in the region $\{\tilde{x}, \tilde{y} \,|\, |\tilde{x} - x| \leqq h_1, |\tilde{y} - y| \leqq h_2\}$. If $(u_0, v_0) \neq (0, 0)$, new tiles are created at $(x, y, 0)$ with intensity $\xi = (\xi_1, \xi_2, 0)$, such that $\sqrt{\xi_1^2 + \xi_2^2} \leqq \xi_{\max}$, where $\xi_{\max}$ is a chosen small parameter. As a result the new values of u_0 and v_0, denoted by $\tilde{u}_0$ and $\tilde{v}_0$, satisfy

$$|\tilde{u}_0| \leqq \xi_{\max}, \qquad |\tilde{v}_0| \leqq \xi_{\max}. \tag{4.1}$$

Periodic boundary conditions were imposed in the following way. For each blob or tile located at (x, y, z) two other imaginary blobs or tiles were added at $(x, y \pm q, z)$. To save computational time, further blobs or tiles were not added, as their contribution to the flow quantities became smaller the further they are from the computational domain.

We restrict ourselves to the domain $0 \leqq x \leqq X_0$, rather than $0 \leqq x \leqq \infty$. Thus we remove any blob or tile whose x-component location exceeds X_0. This is reasonable, since blobs and tiles located far away from the region of interest contribute little to the overall flow. In addition, this procedure economized the cost of computation, for otherwise a large number of computational elements became bunched near $x = X_0$.

5. Convergence. The first convergence proof for vortex methods was given by Hald and Del Prete [24] for the two-dimensional Euler's equations. Convergence for the three-dimensional version of vortex method that was suggested by Beale and Majda, for which spatial derivatives are approximated by finite differences, was given in [4], [5], [2]. For our scheme, in which explicit differentiation is applied to approximate spatial derivatives, convergence was first proved by Beale [3], and then, using a different approach, by Cottet [10]. We quote the theorem appearing in [10], since it applies to a slightly more general case, i.e., the restriction $d \geqq 4$, where d appears in (5.3)-(5.4) below, is removed in [10].

Let us first define for $p \in [1, \infty)$ and $m \geqq 0$ the Sobolev spaces

$$W^{m,p} = \{f, \partial^\alpha f \in L^p(R^n), |\alpha| \leqq m\}$$

and by $\|\cdot\|_{m,p}$ the norm

$$\|f\|_{m,p} = \left(\sum_{0 \leqq |\alpha| \leqq m} \|\partial^\alpha f\|_{0,p}^p \right)^{1/p},$$

and for $p = \infty$ the usual modification.

THEOREM. Convergence in 3D [10]. *Assume that the initial vorticity ξ_0 is smooth enough and that the following conditions hold for the cutoff function ϕ:*

$$\phi \in W^{m,\infty}(R^3) \cap W^{m,1}(R^3) \quad \forall m > 0, \tag{5.1}$$

$$\int_{R^3} \phi(\mathbf{x})\, d\mathbf{x} = 1, \tag{5.2}$$

$$\int_{R^3} \mathbf{x}^\alpha \phi(\mathbf{x})\, d\mathbf{x} = 0, \qquad |\alpha| \leqq d - 1, \tag{5.3}$$

$$\int_{R^3} |\mathbf{x}|^d \phi(\mathbf{x})\, d\mathbf{x} < \infty, \tag{5.4}$$

and that there exist constants C and $\beta > 1$ such that

$$h \leqq C\delta^\beta. \tag{5.5}$$

Then there exists a time τ and a constant C, depending only on ξ_0, such that for h and δ small enough

$$\|\tilde{u} - u\|_{0,p} \leqq C\delta^d, \qquad p \in (3/2, \infty], \quad t \in [0, \tau].$$

We now apply this theorem to our scheme. Using the relation $\phi(r) = f'(r)/4\pi r^2$ derived in [6], we find that

$$\phi(r) = \begin{cases} 0, & r \geqq 1, \\ 15 \cdot (1 - r^2)/8\pi, & r < 1. \end{cases}$$

It is easy to verify that $\phi(r)$ satisfies (5.2) with $d = 2$. In addition, if we choose the cutoff function ϕ to be infinitely smooth, second-order accuracy is achieved. We would now like to view the importance condition (5.1), in the case where the latter is satisfied for finite m only.

The error in vortex methods is usually estimated by bounding the part caused by the regularization of the singular kernel separately, and from the one caused by the discretization of the equations. We therefore write the error in the following form:

$$\mathbf{e} = \tilde{\mathbf{u}} - \mathbf{u} = (\mathbf{u}_\delta - \mathbf{u}) + (\tilde{\mathbf{u}} - \mathbf{u}_\delta) = \mathbf{e}_r + \mathbf{e}_d,$$

where $\mathbf{e}_r$ is the regularization error, caused by replacing the singular kernel K by a smoothed one K_δ, and $\mathbf{e}_d$ is the discretization error.

It was proved in [3] and Lemma 5.5 of [10] that

$$\|\mathbf{e}_r\|_{0,p} \leqq C\delta^d, \qquad p \in (3/2, \infty]$$

for some time $t \in [0, \tau]$, provided that (5.2)-(5.4) hold. In addition, as was shown in [10],

$$\|\mathbf{e}_d\|_{0,p} \leqq Ch^m/\delta^{m-1}, \tag{5.6}$$

in case that (5.1) holds for every $m>0$. A generalization of this theorem for finite m was given in [38, p. 315] for a two-dimensional problem. We must assume, in addition, that

$$\phi \in W^{m-1,\infty}(R^2) \cap W^{m-1,1}(R^2), \qquad m \geqq 3$$

or $\phi \in W^{m-1,\infty}(R^2)$ for $m \geqq 2$ and has compact support. Then for all arbitrarily small $s>0$ there exist a constant C_s, such that

$$\|\tilde{\mathbf{u}}-\mathbf{u}\|_{0,\infty} \leqq C_s \delta^{-s}(\delta^d + h^m/\delta^{m-1}),$$

provided that (5.5) is replaced by $c_2^{-1}\delta^\alpha \leqq h \leqq c_1 \delta^\beta$, with $\alpha \geqq \beta > 1$. Therefore, by choosing α, β appropriately we can balance the regularization error with the discretization error. Similar results were proven in [2] for the three-dimensional vortex methods suggested by Beale and Majda. In our case $\phi \in W^{1,\infty}(R^3)$ and has compact support, and if we could apply similar results to a three-dimensional problem, the discretization error would have been $O(h^2/\delta)$, Therefore, for $\delta = Ch^{2/3}$, the error is at most of order $h^{4/3}$. This can be improved by choosing an infinitely smooth cutoff function.

It was observed in numerical experiments (e.g., [6]) that the formal accuracy of the vortex-blob method might be degraded for a set of radially symmetric test problems. This was understood as loss of accuracy due to the distortion of the initial grid. Beale and Majda [6] suggested to rezone the grid as time evolves. Another way to overcome this difficulty is to use a fixed grid for this set of problems. This method is discussed and analyzed in [20]. It is most probable, however, that the grid is not as much distorted for a flow past a plate, and therefore the vortex-blob method can be used on the Lagrangian grid for this problem. A consideration of a fixed-grid vortex scheme for this problem needs more extensive research.

We turn to the accuracy of the random walk used to model viscosity. It is well known that in two dimensions the random walk approximates the heat equation, though without high accuracy. More accurate error estimates were given in [23] for a one-dimensional heat equation, using a random-walk method with creation of vorticity, i.e.,

$$P\left(\frac{\|\tilde{u}-u\|_{L^2}}{\|u\|_{L^2}} \leqq C_R\left(\frac{\Delta t}{t}+\frac{k}{\sqrt{N}}\right)\right) \geqq 1-\frac{1}{k^2},$$

where

$$C_R = \left(1+\frac{1}{R}\right)\left(1+\frac{1+\sqrt{1/R}}{\sqrt{1+1/R}}\right),$$

N is the number of tiles, k is an arbitrary positive number, Δt is the timestep, and P denotes probability. Note that C_R is a decreasing function of R. A numerical study of the vortex sheet method for the Prandtl equations was done by Puckett [37] together with a spline smoothing of the velocity field. Convergence of this method was demonstrated numerically, with consistency error of order $(h+\xi_{\max})\sqrt{\Delta t/R}$. Here h is the initial spacing in the streamwise direction and $\xi_{\max}$ is the maximal vorticity of newly created sheets.

In [19] we suggest a new way to discretize the viscous term. The idea is to convolve the vorticity with a cutoff function, and approximate the Laplacian of the vorticity by the convolution of the Laplacian of the cutoff function against the vorticity. Another deterministic method was suggested by Degond and Mas-Gallic in [16].

6. Numerical results. We must specify the following parameters for our numerical scheme. The initial grid, with spacing h_1, h_2, the timestep Δt, the maximum allowed intensity of a newly created tile, $\xi_{\max}$, are parameters to be chosen. In addition, the cutoff δ, the thickness of the layer for which the Prandtl equations are used z_0, and the physical domain $0 \leqq x \leqq X_0$, in which we keep track of the motion of the computational elements, must be specified. We set $X_0 = 1.5$ (as in [11]). We picked $z_0 = C\sqrt{2\Delta t/R}$, with $C = 1.5$, $\sqrt{2\Delta t/R}$ being the standard deviation of the random walk. We made this choice for z_0 to ensure that a tile, located in the layer $0 \leqq z \leqq z_0$, will have a high probability of moving out of the tile layer in a few timesteps, and will then turn into a blob. We picked $U = L = 1$ and the viscosity $\nu = 10^{-4}$, so that the Reynolds number $R = UL/\nu = 10^4$. This value was high enough to show the three-dimensional effects and the transition to turbulence, as was also observed in the experiments in [25]. Note that the local Reynolds number $R_x = Ux/\nu$ depends on x but R does not. Following Chorin [11] we picked h_1/π as the cutoff δ. This is in agreement with the condition in the convergence theorem in § 5, that the cutoff δ should be larger than the typical distance between neighboring particles, the latter being of order $1/\sqrt{R}$ in our problem.

After fixing X_0, z_0, and choosing δ, we had to pick the initial spacing h_1, h_2, $\xi_{\max}$, and the timestep Δt. To do this, we first ran the two-dimensional problem, in which the independent variables are x, z, and whose steady-state solution is analytically known to be the Blasius solution. We found out, as was also pointed out in [40], that h_1 and $\xi_{\max}$ have primary importance, since they control the number of newly created sheets. The latter determines the number of blobs, and therefore the number of computational elements. If larger numbers of computational elements are used, the error in both interior and exterior regions decreases. We tried the following choices for h_1, Δt, and $\xi_{\max}$:

(a) $h_1 = \Delta t = 0.20$, $\xi_{\max} = 0.1$.
(b) $h_1 = \Delta t = 0.15$, $\xi_{\max} = 0.075$.
(c) $h_1 = \Delta t = 0.10$, $\xi_{\max} = 0.050$.

For these sets of parameters we checked the drag, given by the following formula (see e.g., [41], [11]):

$$D(x_0) = \int_0^\infty u(x_0, z)(U_\infty - u(x_0, z))\,dz, \tag{6.1}$$

and compared it with the Blasius drag $D_0 = 0.6641\sqrt{x_0/R}$. The integral in equation (6.1) was discretized by the trapezoidal rule

$$D_{\text{com}} = \sum_{i=0}^{m} c_i u(x_0, i\,\Delta z)(U_\infty - u(x_0, i\,\Delta z))\,\Delta z,$$

where $c_0 = c_m = 0.5$, and $c_i = 1$, for $1 \leqq j \leqq m-1$. Here $m = z_{\max}/\Delta z$, where $z_{\max}$ is the maximal z, for which computational points were found in the region $|x - x_0| \leqq h_1$, and Δz was chosen to be 0.004. The relative error in the drag $|(D(x_0) - D_0(x_0))/D_0(x_0)|$ for $x_0 = 1$ is given in Table 1. In addition, to measure the intensity of the noise from the statical process, we averaged the computed drag every 10 iterations, i.e.,

$$D_{\text{avg}} = \frac{1}{10}\sum_{n=0}^{9} D_{\text{com}}(t - n\,\Delta t) \tag{6.2}$$

and calculated the variance of the instantaneous drag from the averaged one. The

TABLE 1

Grid	Relative drag error	Variance	No. of sheets	No. of blobs	Time
(a)	0.52	0.061	95	76	3 min.
(b)	0.21	0.024	139	151	13 min.
(c)	0.14	0.016	344	415	127 min.

variance of the drag is given by the following formula:

$$V(D)=E(|D-E(D)|^2), \tag{6.3}$$

where $E(X)$ is the expected value of a random variable X. We approximated the expected values $E(X)$ in (6.3) X_{avg}, where the average is computed as in (6.2). We would like to reduce the statistical noise, and therefore to decrease the variance by choosing the appropriate parameters. The results for the drag and variance in the two-dimensional problem are given in Table 1. The total computational time on a VAX-VMS computer is given in this table as well.

The dominant term of the error, as seen by the convergence analysis, is the one due to random walk. This error is of order $1/\sqrt{n}$, where n is the number of sheets or blobs. When we refined grid (a) to (b), we approximately doubled the number of computational elements, so that the error should decrease by a factor of $\sqrt{2}$. The computational factor is found to be bigger than two. If we look at grid (c) compared to (b), the number of sheets or blobs was increased by a factor of three approximately, so that the error should decrease by a factor of $\sqrt{3}$, which is approximately 1.7. The computed factor was found to be 1.5. From Table 1 we can learn that the finer the grid, the smaller the relative error in the drag, and the smaller the variance. In addition, much more time is required for grid (c) than for grid (b). To make our computations affordable for a longer time in the three-dimensional problem, we chose the three-dimensional grid (b). We also had to specify h_2 for three-dimensional problems. We chose $h_2=q/4$ for grid (a), $h_2=q/6$ for grid (b), and $h_2=q/8$ for grid (c).

We examined the instability of the Blasius solution for high Reynolds numbers in a three-dimensional problem. This was done as follows (see [11]). For $0\leqq t\leqq T=1$ we approximated the Prandtl equations, whose steady-state is the Blasius solution, using only tiles. Note that instability does not occur for the Prandtl equations, whereas it might occur for the Navier-Stokes equation. The numerical solution converges to the Blasius solution as $t\to\infty$ and h_1, $h_2\to 0$, where h_1, h_2 is the size of the initial grid. We used the results of this scheme at $T=1$ to be the initial conditions for the Navier-Stokes equations. Instability for the Navier-Stokes equations is shown, i.e., small perturbations in the Blasius solution cause large changes in the solution. We perturbed the Blasius solution by choosing the following initial condition at infinity:

$$\mathbf{u}(x,y,\infty,T)=\begin{cases}(U_\infty,A,0) & \text{for } \frac{1}{4}q<y<\frac{3}{4}q,\\ (u_\infty,0,0) & \text{elsewhere,}\end{cases}$$

where $A=10^{-3}$. After $T=1$ we used the scheme described in §§ 3 and 4, in which tiles and blobs are present, and therefore instability might occur.

We display all the results at $t=22.5$. Velocity and vorticity are shown in the following two-dimensional planes: (a) $y=\frac{1}{2}q$, which describes the flow quantities as a function of x and z; (b) at the two planes $x=1$, 1.4, which shows the velocity and vorticity as a function of y and z. Note that as x increases the more apparent are the three-dimensional features, i.e., the dependence on y and the transition to turbulence. This happens since the local Reynolds number $R_x = Ux/L$ increases for larger x.

In Figs. 1-3 we display velocity components computed at a regular mesh. Figure 1 shows the x, z components of the velocity at $y = q/2$. In Figs. 2-3 the y, z components of the velocity at $x = 1$, 1.4, respectively, are displayed. These figures, as well as other

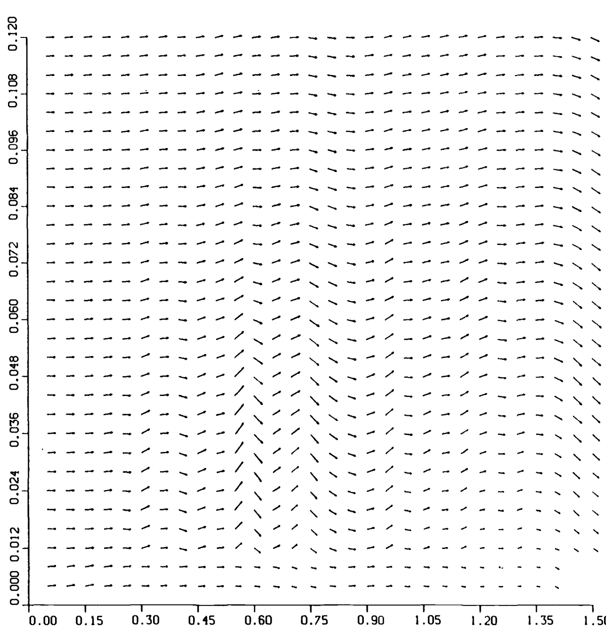


FIG. 1. *Velocity field in the x, z plane for y = q/2.*

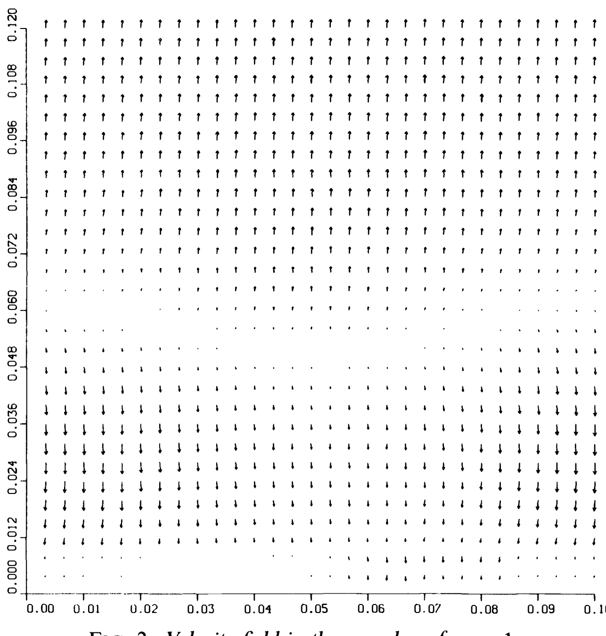


FIG. 2. *Velocity field in the y, z plane for x = 1.*

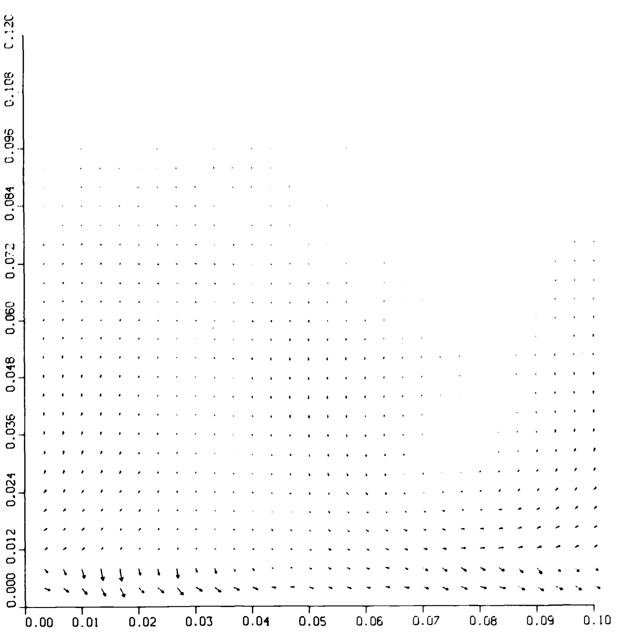


FIG. 3. *Velocity field in the y, z plane for $x = 1.4$.*

figures represented for fixed x, show the three-dimensional features of the flow, i.e., the dependence on y. This is in accordance with results appearing in [35] and [36]; in the latter numerical results were performed for a periodic problem in both x and y. They indicate the three-dimensional character of secondary instability, which is consistent with the idea that turbulence is intrinsically three-dimensional. Vorticity is represented in the Lagrangian computational grid points in Figs. 4-6. In Fig. 4 the x, z components of vorticity at $y = q/2$ is displayed. We can see that for larger x the intensity of the vorticity increases, which is one of the features of transition to turbulence, i.e., vorticity is no longer preserved in the Lagrangian system as it is in a two-dimensional problem.

In Figs. 5-6 we show the y, z components of vorticity at $x = 1$, 1.4, respectively. Note that for larger x the vorticity is no longer directed in one direction. This is in agreement with the results in [25], which indicate the appearance of small hairpins as the flow develops in the streamwise direction. Figures 7-9 show contours of the z-component of vorticity. These figures indicate that for larger x small scale phenomena appear. Figures 10-12 show contours of the y-component of vorticity, in which the results are similar to those of the z-component of vorticity.

In Tables 2 and 3 we show the averaged drag (multiplied by a factor of 100), the thickness of the boundary layer and the drag variance for $y = q/2$, $x = 1$ and $y = q/2$, $x = 1.4$, respectively, and for different time levels. The averaged drag and variance were calculated in the same way as for the two-dimensional problem. The boundary layer thickness is given by

$$\text{Thickness}\,(x, y) = \int_0^\infty (U_\infty - u(x, y, z))\,dz,$$

and is computed for $y = q/2$. The integral is discretized by the trapezoidal rule.

We notice that the drag grows until it reaches a certain level at about $t = 6$ and then stabilizes. As time progresses the variance at $x = 1$ also stabilizes and stays at a

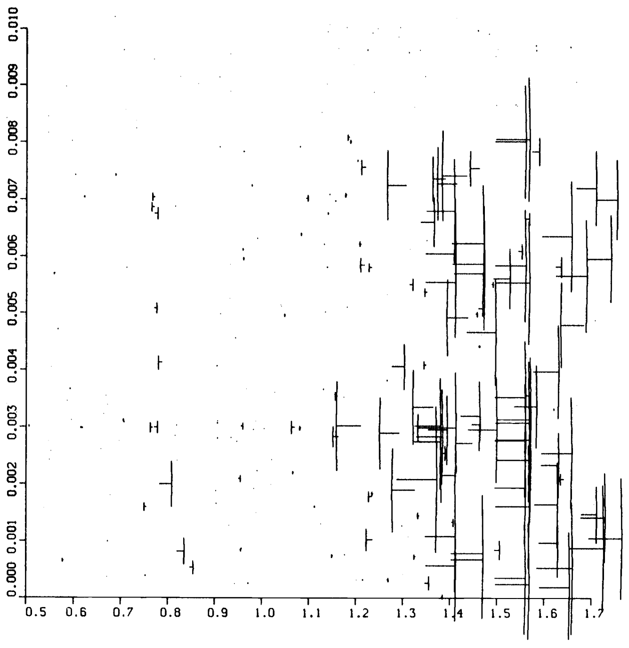


FIG. 4. *Vorticity in the x, z plane for $y = q/2$.*

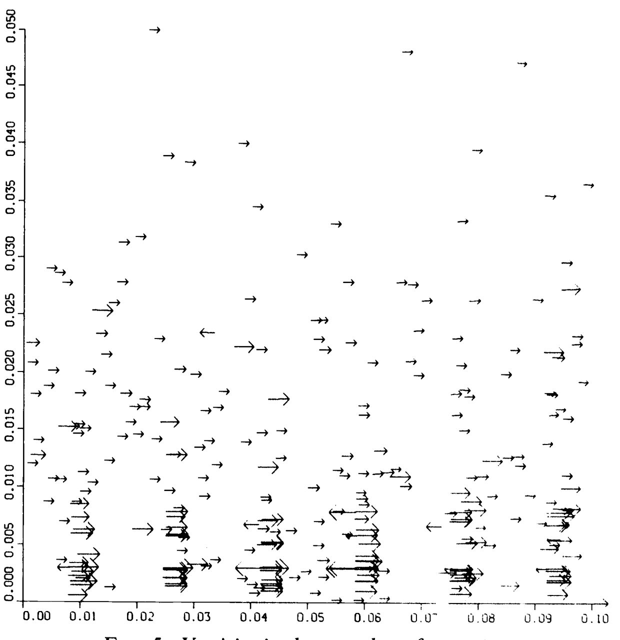


FIG. 5. *Vorticity in the y, z plane for $x = 1$.*

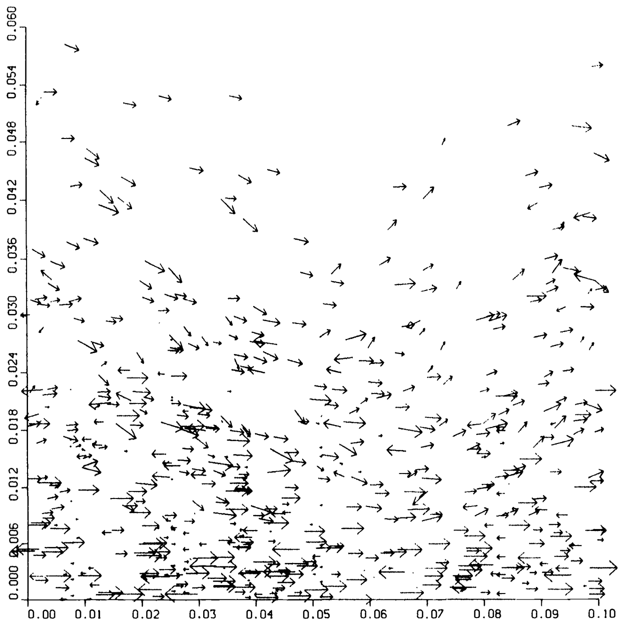


FIG. 6. *Vorticity in the y, z plane for $x = 1.4$.*

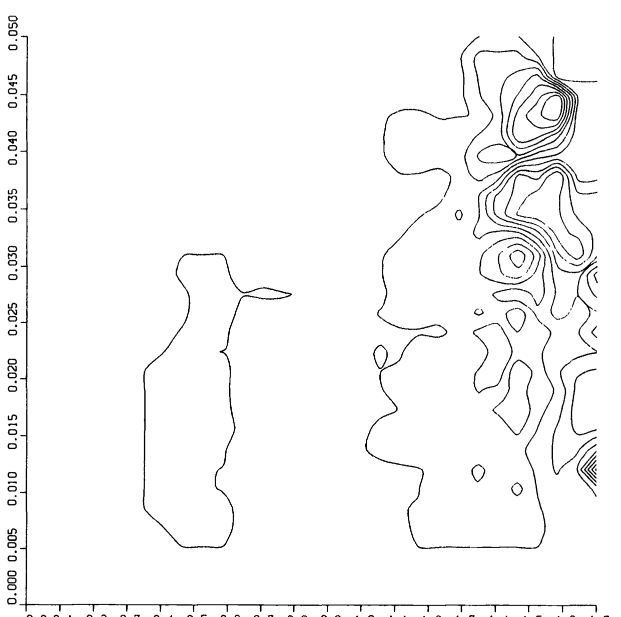


FIG. 7. *Contours of the z component of vorticity in the x, z plane for $y = q/2$.*

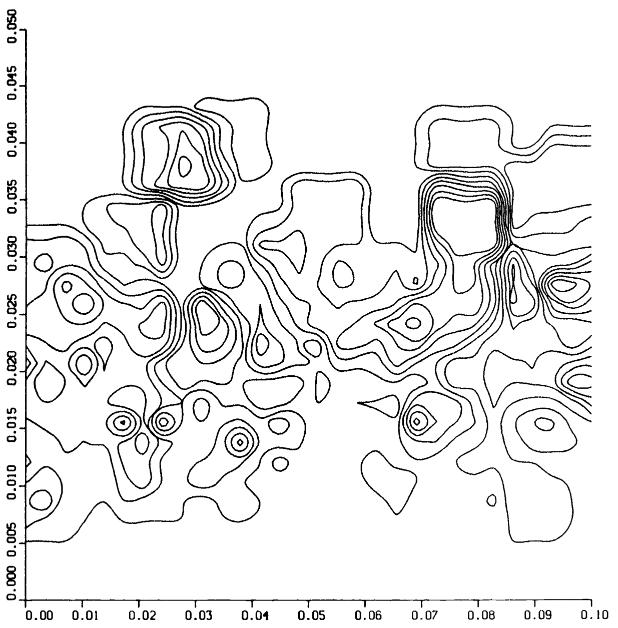


FIG. 8. *Contours of the z component of vorticity in the y, z plane for* $x = 1$.

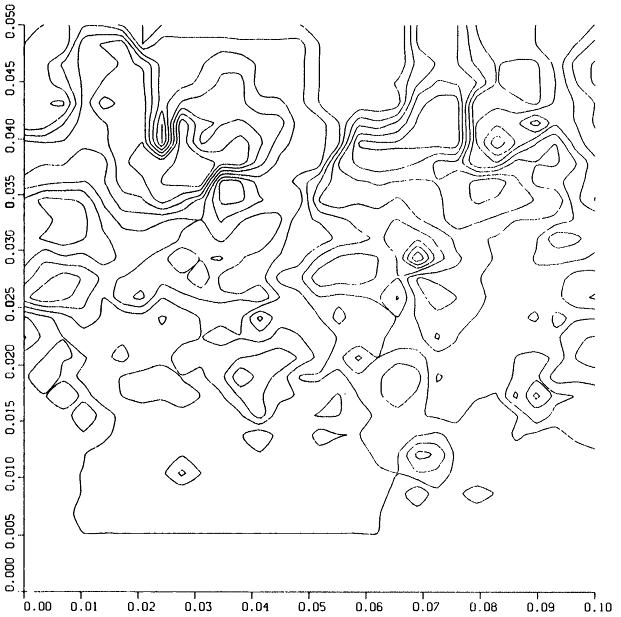


FIG. 9. *Contours of the z component of vorticity in the y, z plane for* $x = 1.4$.

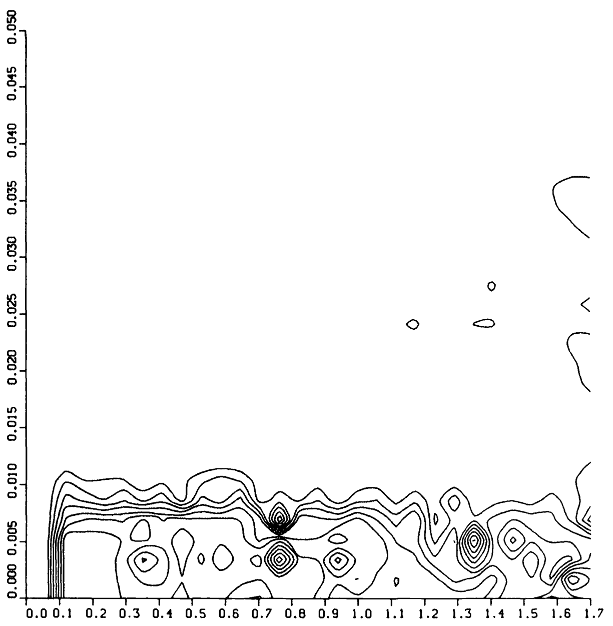


FIG. 10. *Contours of the y component of vorticity in the x, z plane for* $y = q/2$.

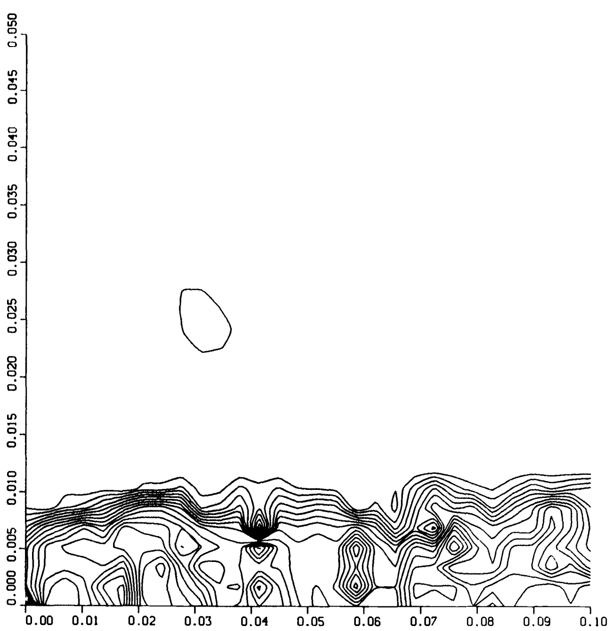


FIG. 11. *Contours of the y component of vorticity in the y, z plane for* $x = 1$.

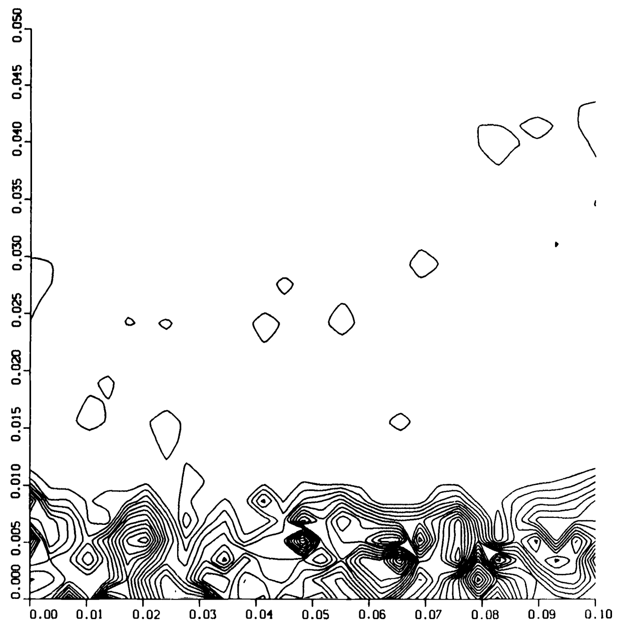


FIG. 12. *Contours of the y component of vorticity in the y, z plane for* $x = 1.4$.

TABLE 2
Averaged drag, thickness, and variance at $x = 1$.

	$t=1.5$	$t=3$	$t=6$	$t=12$	$t=22.5$
Drag	0.2713	0.4647	0.5262	0.5245	0.5718
Thickness	56.340	81.710	80.510	87.820	100.491
Variance	0.0376	0.0283	0.0128	0.0098	0.0080

TABLE 3
Averaged drag, thickness, and variance at $x = 1.4$.

	$t=1.5$	$t=3$	$t=6$	$t=12$	$t=22.5$
Drag	0.2582	0.4731	0.5221	0.5450	0.4194
Thickness	57.340	92.170	107.05	136.17	101.776
Variance	0.0348	0.0188	0.0194	0.0400	0.0527

level of about 0.01. However, the variance at $x = 1.4$ changes in time and starts to grow as time progresses. Therefore, we notice that the flow downstream is changing more rapidly in time. The thickness of the boundary layer increases as time progresses until it reaches a certain level and then starts to oscillate rapidly, especially at $x = 1.4$. If we compare the thickness of the boundary layer at $x = 1$ to the one at $x = 1.4$, we notice that this quantity grows as we proceed in the downstream direction.

In Figs. 13-15 we show the u component of the velocity as a function of the similarity variables $\eta = y\sqrt{U_\infty/x}$. In every graph we display the velocity at both $x = 1$ and $x = 1.4$. Figures 13, 14, and 15 correspond to $t = 6$, 12, and 22.5, respectively. We

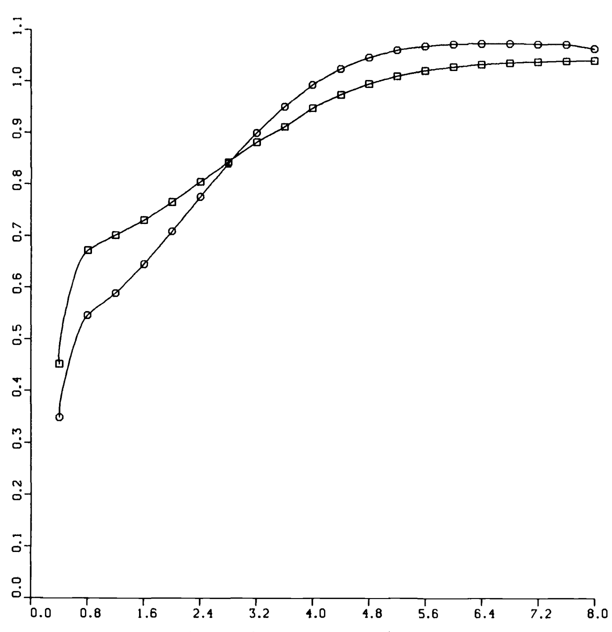


FIG. 13. *u-component of the velocity at* $t = 6$, ◇— *at* $x = 1$, ○— *at* $x = 1.4$.

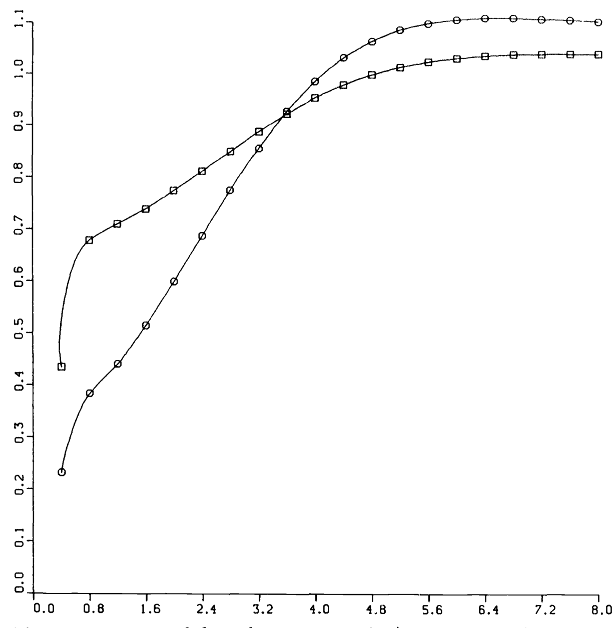


FIG. 14. *u-component of the velocity at* $t = 12$, ◇— *at* $x = 1$, ○— *at* $x = 1.4$.

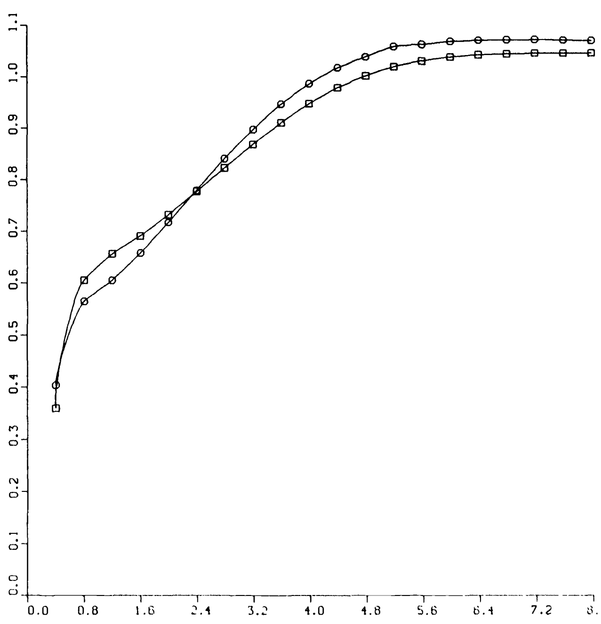


FIG. 15. *u-component of the velocity at* $t = 22.5$, ◇— *at* $x = 1$, ○— *at* $x = 1.4$.

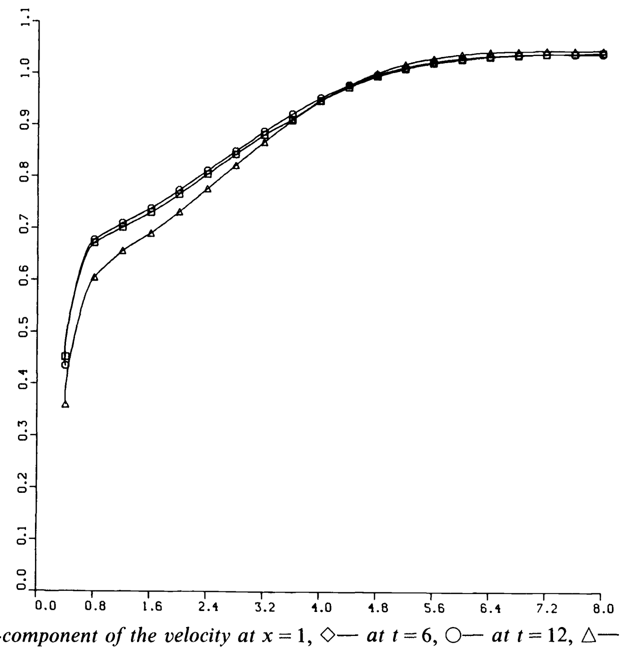


FIG. 16. *u-component of the velocity at* $x = 1$, ◇— *at* $t = 6$, ○— *at* $t = 12$, △— *at* $t = 22.5$.

can learn from these figures that the velocity varies with x, whereas in Blasius's solution the solution is a function of η only. In Figs. 16–17 we show the same quantity u as a function of η, but now several time levels are shown in Fig. 16 for $x = 1$ and in Fig. 17 for $x = 1.4$. We can learn from Figs. 16–17 that the flow quantities oscillate more in time when we proceed in the streamwise direction. This is explained by the growth of the local Reynolds number.

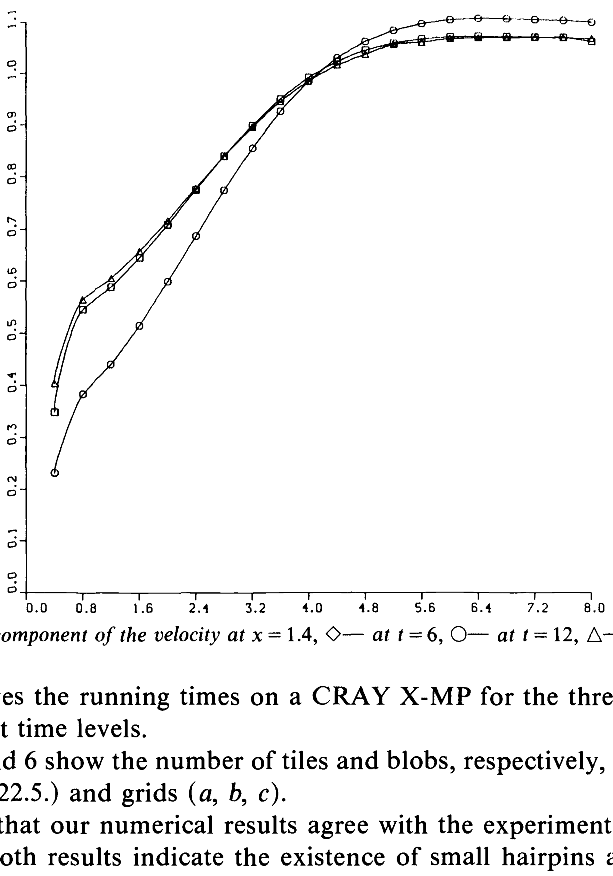


FIG. 17. *u-component of the velocity at* $x = 1.4$, ◇— *at* $t = 6$, ○— *at* $t = 12$, △— *at* $t = 22.5$.

Table 4 gives the running times on a CRAY X-MP for the three different grids, and for different time levels.

Tables 5 and 6 show the number of tiles and blobs, respectively, for various times ($t = 3, 6, 9, 12, 22.5.$) and grids (a, b, c).

We found that our numerical results agree with the experimental results of [25] in a way that both results indicate the existence of small hairpins at high Reynolds numbers. Note that in other experiments horseshoe vortices rather than small hairpins were found. As was explained in [25], the reason for the different results was that the experimental techniques of other investigators limited the results to low Reynolds numbers.

7. Conclusions. The three-dimensional version of vortex methods used here were capable of resolving the three-dimensionality of the flow and the transition to tur-

TABLE 4
Total computational time to reach $t = 3, 6, 9, 12, 22.5$.

Grid	$t=3$	$t=6$	$t=9$	$t=12$	$t=22.5$
(a)	0.5 min	1 min	2 min	3 min	6 min
(b)	7 min	25 min	43 min	1 h 7 min	2 h 13 min
(c)	2 h 30 min				

TABLE 5
Number of tiles.

Grid	$t=3$	$t=6$	$t=9$	$t=12$	$t=22.5$
(a)	263	323	344	360	348
(b)	1,080	962	1,000	926	1,098
(c)	4,548				

TABLE 6
Number of blobs.

grid	$t=3$	$t=6$	$t=9$	$t=12$	$t=22.5$
(a)	205	213	187	183	222
(b)	1,051	915	830	947	987
(c)	6,948				

bulence. Away from the plate, we used a three-dimensional blob method, which is a natural extension of two-dimensional vortex methods. These methods can have high spatial accuracy, and they involve no elaborate calculation. Near the plate, the tile method approximates a thin boundary layer, and is a straightforward extension of the two-dimensional sheet method. Therefore the two-dimensional and the three-dimensional problems can be similarly treated numerically.

Acknowledgments. I thank Professor Alexandre Chorin for many helpful discussions, Dr. Scott Baden for vectorizing the code, and the referees for their valuable comments.

REFERENCES

[1] C. ANDERSON, *Vortex methods for flows with variable density*, Ph.D. thesis, University of California, Berkeley, CA, 1983.

[2] C. ANDERSON AND C. GREENGARD, *On vortex methods*, SIAM J. Numer. Anal., 22 (1985), pp. 413-440.

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

[4] T. BEALE AND A. MAJDA, *Vortex methods* I: *Convergence in three dimensions*, Math. Comp., 39 (1982), pp. 1-27.

[5] ———, *Vortex methods* II: *Higher order accuracy in two and three dimensions*, Math. Comp., 39 (1982), pp. 29-52.

[6] ———, *High order accurate vortex methods with explicit velocity kernels*, J. Comput. Phys., 58 (1985), pp. 188-208.

[7] J. BELL, P. COLELLA, AND H. GLAZ, *A second order projection method for the incompressible Navier-Stokes equations*, J. Comput. Phys., in press, UCRL-98225, Lawrence Livermore Laboratory, Livermore, CA, 1988.

[8] D. J. BENNEY, *A nonlinear theory for oscillations in a parallel flow*, J. Fluid Mech., 10 (1961), pp. 209-236.

[9] D. J. BENNEY AND C. C. LIN, *On the secondary motion induced by oscillations in a shear flow*, Phys. Fluids, 3 (1960), pp. 656-657.

[10] G. H. COTTET, *A new approach for the analysis of vortex methods in two and three dimensions*, Ann. Inst. H. Poincaré, Analyse non lineaire.

[11] A. J. CHORIN, *Vortex models and boundary layer instability*, SIAM J. Sci. Statist. Comput., 1 (1980), pp. 1-21.

[12] A. J. CHORIN, *Numerical study of slightly viscious flow*, J. Fluid Mech., 57 (1973), pp. 785-796.

[13] ———, *Vortex sheet approximation of boundary layers*, J. Comput. Phys., 27 (1978), pp. 428-442.

[14] ———, *Scaling laws in the vortex lattice model*, Comm. Math. Phys., 114 (1988), pp. 167-176.

[15] A. J. CHORIN AND J. E. MARSDEN, *A Mathematical Introduction to Fluid Mechanics*, Springer-Verlag, Berlin, New York, 1979.

[16] P. DEGOND AND S. MAS-GALLIC, *The weighted particle method for convection-diffusion equations, Part* I: *The case of isotropic viscosity*, Math. Comp., in press.

[17] D. FISHELOV, *Spectral methods for the small disturbance equation of transonic flows*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 232-251.

[18] ———, *The spectrum and the stability of the Chebyshev collocation operator for transonic flow*, Math. Comp., 51 (1988), pp. 559-579.

[19] ———, *A new vortex scheme for viscous flows*, LBL-25556, Lawrence Berkeley Laboratory, University of California, Berkeley, CA, July 1988; J. Comput. Phys., 85 (1990), pp. 211-224.

[20] ———, *A fixed grid for vortex methods*, Math. Comp., submitted, LBL-27489, Lawrence Berkeley Laboratory, University of California, Berkeley, CA, February 1989.

[21] D. GOTTLIEB, *Strang type difference schemes for multidimensional problems*, SIAM J. Numer. Anal., 9 (1972), pp. 650-661.

[22] C. GREENGARD, *The core spreading vortex method approximates the wrong equation*, J. Comp. Phys., 61 (1985), pp. 345-348.

[23] O. H. HALD, *Convergence of a random method with creation of vorticity*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1373-1386.

[24] O. HALD AND V. DEL PRETE, *Convergence of vortex methods for Euler's equations*, Math. Comp., 32 (1978), pp. 791-809.

[25] M. R. HEAD AND P. BANDYAPODHYAY, *New aspects of turbulent boundary layer structure*, J. Fluid Mech., 107 (1981), pp. 297-338.

[26] R. KERR, *Reconnection in three-dimensional vortex motion*, to appear.

[27] J. KIM AND P. MOIN, *The structure of the vorticity field in turbulent channel flow, Part* 2. *Study of ensemble-averaged fields*, J. Fluid Mech., 162 (1986), pp. 339-363.

[28] R. B. KINNEY AND M. A. PAOLINO, *Flow transient near the leading edge of flat plate moving through a viscous fluid*, J. Appl. Mech., 41 (1974), pp. 919-924.

[29] R. B. KINNEY AND Z. M. CIELAK, *Analysis of unsteady viscous flow past an airfoil*: *Part* 1—*theoretical development*, AIAA J., 15 (1977), pp. 1712-1717.

[30] P. S. KLEBANOFF, K. D. TIDSTROM, AND L. M. SARGENT, *The three-dimensional nature of the boundary-layer instability*, J. Fluid Mech., 12 (1962), pp. 1-34.

[31] S. J. KLINE, W. C. REYNOLDS, F. A. SCHRAUBAND, AND P. W. RUNSTADLER, *The structure of turbulent boundary layers*, J. Fluid Mech. 30 (1967), pp. 741-773.

[32] A. LEONARD, *Computing three-dimensional incompressible flows with vortex elements*, Ann. Rev. Fluid Mech., 17 (1985), pp. 523-559.

[33] ———, *Vortex simulation of three dimensional spotlike disturbances in laminar boundary layer*, in Turbulent Shear Flows, 2 (1980), Springer-Verlag, Berlin, Heidelberg, 1980, pp. 67-77.

[34] ———, *Vortex methods for flow simulation*, J. Comput. Phys., 37 (1980), pp. 289-335.

[35] S. A. ORSZAG AND L. C. KELLS, *Transition to turbulence in plane Poiseuille and plane Couette flow*, J. Fluid Mech., 96 (1980), pp. 159-205.

[36] S. A. ORSZAG AND A. T. PATERA, *Secondary instablity of wall-bounded shear flows*, J. Fluid Mech., 128 (1983), pp. 347-385.

[37] E. G. PUCKETT, *A study of the vortex sheet method and its rate of convergence*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 298-327.

[38] P. A. RAVIART, *An analysis of particle methods*, in Numerical methods in Fluid Dynamics, F. Brezzi, ed., Lecture Notes in Mathematics, 1127, Springer-Verlag, Berlin, 1985.

[39] L. A. SEGEL, *Mathematics Applied to Continuum Mechanics*, Macmillan, New York, 1977.

[40] J. A. SETHIAN AND A. F. GHONEIM, *Validation study of vortex methods*, J. Comput. Phys., 74 (1988), pp. 283-317.

[41] H. SHLICHTING, *Boundary Layer Theory*, McGraw-Hill, New York, 1960.

[42] E. SIGGIA, *Collapse and amplification of a vortex filament*, Phys. Fluids, 28 (1985), pp. 794-804.

[43] R. A. SCHMALL AND R. B. KINNEY, *Numerical study of unsteady viscous flow past a lifting plate*, AIAA J., 12 (1974), pp. 1566-1573.

[44] VAN DER VEGT, *A variationally optimized vortex algorithm for three-dimensional flows around solid bodies*, Ph.D. thesis, University of Delft, Delft, the Netherlands.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 3, pp. 425-449, May 1990

ADAPTATION OF A TWO-POINT BOUNDARY VALUE PROBLEM SOLVER TO A VECTOR-MULTIPROCESSOR ENVIRONMENT*

S. J. WRIGHT† AND V. PEREYRA‡

Abstract. Systems of linear equations arising from finite-difference discretization of two-point boundary value problems have coefficient matrices that are sparse, with most or all of the nonzeros clustered in blocks near the main diagonal. Some efficiently vectorizable algorithms for factorizing these types of matrices and solving the corresponding linear systems are described. The relative effectiveness of the different algorithms varies according to the distribution of initial, final, and coupled end conditions. The techniques described can be extended to handle linear systems arising from other methods for two-point boundary value problems, such as multiple shooting and collocation. An application to seismic ray tracing is discussed.

Key words. two-point boundary value problems, finite-difference methods, vector processors, seismic ray tracing

AMS(MOS) subject classifications. 34B10, 65F05, 65L10, 86-08

1. Introduction. In this paper we describe new algorithms for solving a sparse linear system of equations that arises in global discretization schemes for solving two-point boundary value problems of the general form

$$z'(t)=f(t,z), \qquad a \leqq t \leqq b, \qquad g(z(a),z(b))=0. \tag{1.1}$$

Here $f, z, g \in R^m$, $t \in R$. These discretization schemes lead to a system of nonlinear equations whose solution is a discrete approximation to the true solution of (1.1). If Newton's method (or some variant) is used to solve this nonlinear system, it is known that for the "obvious" orderings of the equations and unknowns, the Jacobian will have most of its nonzero elements clustered about the main diagonal (see [2], [7], [8]-[10]). To find the Newton correction it is necessary to solve a linear system for which the Jacobian is the coefficient matrix. This operation is carried out repeatedly and is often the most time-consuming part of the solution process.

The particular linear systems that the algorithms of this paper are intended to solve are those arising from the PASVAR series of codes (see [9], [10] and appropriate sections of the IMSL, NAG and Harwell computer software libraries), but they also have other applications. The basis of the PASVAR algorithms is a trapezoidal-rule discretization of (1.1), with deferred corrections used to enhance the solution accuracy. Similar systems arise in algorithms based on the one-step formulae described in Cash [2]. Although we do not deal with linear systems arising from multistep, multiple shooting, and collocation methods here, we note that linear equation solvers analogous to those discussed in § 3 could also be devised for these methods.

* Received by the editors November 9, 1987; accepted for publication (in revised form) May 1, 1989. This research was performed under National Science Foundation Small Business Innovation Research award #8619607. Any opinions, findings, conclusions, and recommendations are those of the authors and do not necessarily reflect the views of the National Science Foundation.

† Mathematics Department, North Carolina State University, Raleigh, North Carolina 27695. The research of this author was partially supported by National Science Foundation grants ASC-8714009, DMS-8619903, and DMS-880033P and Air Force Office of Scientific Research grant AFOSR-ISSA-870092.

‡ Weidlinger Associates, 4410 El Camino Real, Suite 110, Los Angeles, California 94022.

For the trapezoidal-rule discretization of (1.1), a mesh $\{t_1, t_2, \cdots, t_n\}$ is chosen such that

$$a = t_1 < t_2 < \cdots < t_n = b,$$

and the differential equation is replaced by a system of algebraic equations

$$s_{i-1}(z_{i-1}, z_i) = z_i - z_{i-1} - \frac{h_i}{2}[f(t_i, z_i) + f(t_{i+1}, z_{i+1})] = 0, \qquad i = 2, \cdots, n \tag{1.2a}$$

where $h_i = t_i - t_{i-1}$ and $z_i \approx z(t_i)$. Often, the boundary conditions in (1.1) are well structured and can be separated into initial, coupled, and final conditions as follows:

$$\begin{aligned} &g_1(z(a)) = 0 && (g_1 \in R^p), \\ &g_2(z(a), z(b)) = 0 && (g_2 \in R^r), \\ &g_3(z(b)) = 0 && (g_3 \in R^q). \end{aligned} \tag{1.2b}$$

Here $p + q + r = m$. To complete the algebraic equations for the discrete solution $z_1, \cdots, z_n$, we replace $z(a)$ by z_1 and $z(b)$ by z_n in (1.2b). A modified Newton's method is used to solve these equations.

Depending on the ordering of the equations in (1.2) and the unknowns $z_1, \cdots, z_n$, the Jacobian is usually structured in one of the following ways:

(1) Block tridiagonal, where the off-diagonal blocks have some zero rows, and the lower left block is partially filled if $r > 0$ (see Fig. 1). The ordering of equations here is

$$(g_1, s_1, s_2, \cdots, s_{n-1}, g_2, g_3),$$

and the variables are ordered as

$$(z_1, \cdots, z_n).$$

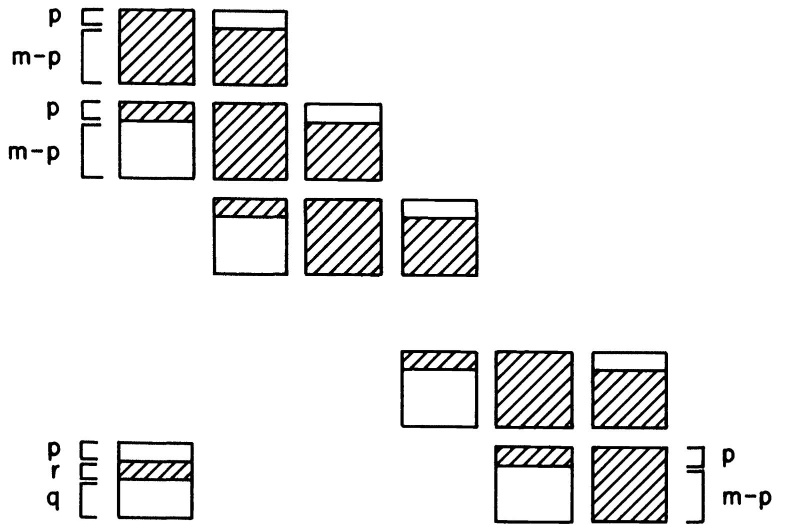


FIG. 1. *"Block-tridiagonal" structure used by method of* § 2. *Shaded areas indicate possible nonzeros.*

(2) Block upper-bidiagonal, with a partially filled block in the lower left corner (see Fig. 2). The ordering of the unknowns here is the same as in (1), but the equations are ordered as follows:

$$(s_1, \cdots, s_{n-1}, g_1, g_2, g_3).$$

The total dimension of the matrix is (mn). In this paper we are interested in the commonly occurring situation in which n is substantially larger than m, corresponding to a fine mesh with a system of differential equations of relatively small dimension.

Existing codes that handle the case of separated end conditions ($r = 0$) [3], [7], [10] use the ordering of Fig. 1. Partial pivoting, involving both row and column swaps at different stages, is used during the factorization. No fill-in occurs, and it is shown in [7] that this pivot strategy ensures a valid factorization is produced whenever the matrix is nonsingular. The algorithm of [10] (to be discussed in § 2) can also handle the case of coupled end conditions. Fill-in of approximately rn elements in the last row of blocks in the L factor takes place. Elements in these blocks are not considered as possible pivots until the final few steps of the elimination. This limitation on the pivoting means that there are some nonsingular matrices for which this algorithm fails to produce a valid factorization (see the example in the Appendix). However, the analysis of Keller [8] can be applied to show that, when the matrix is a Jacobian that arises from a one-step discretization scheme, the algorithm will work when the mesh is sufficiently fine. This is shown in the Appendix.

Some disadvantages become clear when this method is implemented in a vector-processing environment. Since it handles only a small number of blocks at a time (proceeding sequentially down the diagonal in Fig. 1), vectorizable loops in the code involve m or fewer elements.

The algorithms to be discussed here, which use the second ordering, start by factorizing the first $(n-1)$ diagonal blocks concurrently. This allows vectorizable loops of length $(n-1)$ to appear in the code. Similar loops arise in the factorization of the above-diagonal blocks. Since we are assuming $n > m$ here, there is clearly greater potential for speedup in the new algorithms. However, there is a trade-off: the new methods tend to produce more fill-in and have higher operation counts, and hence are unsuitable for use in a scalar environment. A blocked ordering similar to that of Fig. 2 was used originally in the first version of PASVAR [9], although of course it was not then solved by a vectorized method.

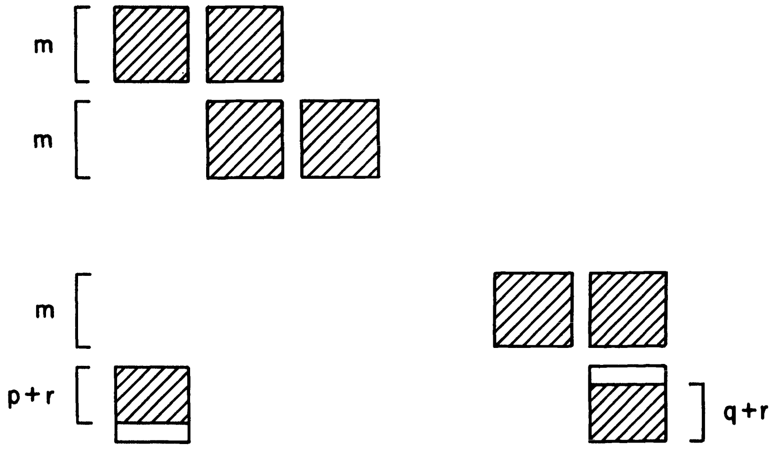


FIG. 2. "*Nearly block-bidiagonal*" *structure used by methods of* § 3.

Since these algorithms essentially perform a block factorization, with row pivoting allowed within blocks but not between blocks, they cannot factorize every nonsingular matrix of the form of Fig. 2, except in the case $p+r=0$. The analysis of the Appendix can be applied again, however, to show that they will succeed if the mesh is sufficiently fine. Numerical experience suggests that in low precision environments (e.g., 4-byte floating point wordlength), the new algorithms may not be as stable as the algorithm in [10]. This is not surprising, since the $p+r$ rows at the bottom of the matrix are always excluded from consideration as pivots, rather than just r rows in [10]. It may therefore be wise to retain the existing codes as a "backup" in case the new methods fail.

The new algorithms are discussed in § 3. In § 4, we present some timing comparisons between the new methods and the existing codes on the Alliant FX/8 vector multiprocessor, the CRAY X-MP, and the CRAY-2. Application of the resulting vectorized code to the problem of seismic ray tracing is discussed in the final section, and some results from the CRAY-2 are given.

In [5], Goldmann considers vectorization of the multiple shooting method. He notes that in many applications it is possible to vectorize the process of evaluating derivatives of f at different points. That is, the (j, k) elements of the $m \times m$ blocks $\partial f/\partial z(t_i, z_i)$ can be evaluated concurrently for $i=1, \cdots, n$. Since this aspect of the overall method is problem-dependent, we do not discuss it further. Goldmann also considers a partially vectorizable method for solution of a linear system similar to that of Fig. 2, but it depends strongly on the fact that the above-diagonal blocks are multiples of the $m \times m$ identity matrix, and hence cannot be applied here.

2. The Routines DECOMP and SOLVE.

The routine DECOMP in the PASVAR codes takes the Jacobian matrix A (as ordered in Fig. 1) and produces two factors L and U such that

$$LU = PAQ^T, \tag{2.1}$$

where P and Q are permutation matrices. The matrix U is only block upper triangular and so (2.1) is not a true LU factorization; however, linear systems involving L and U can be easily solved. The row and column interchanges (which are accounted for in P and Q, respectively), are chosen so that there is no fill-in near the main diagonal. If $r>0$, then r dense rows will appear at the bottom of the L factor (see Fig. 3).

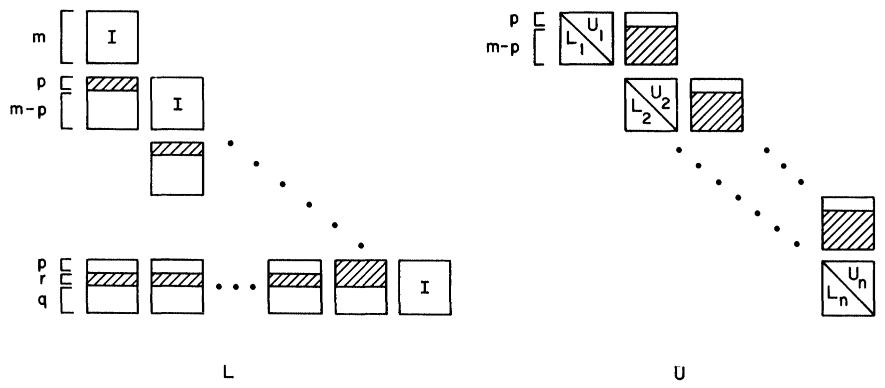


FIG. 3. *Structure of the factors produced by* DECOMP. *(L_i and U_i are the L and U factors of the (i, i) block.)*

The partial pivoting strategy needs some explanation. For the factorization of the first p rows of the matrix, column interchanges are used to ensure that the pivot element has the largest magnitude of any element in its row. Clearly, only columns 1 through m are candidates for the pivot column, since all other columns have zeros in rows 1 to p, and no fill-in occurs. For the factorization of the remaining rows in the $(1,1)$ block, row interchanges involving rows $p+1$ through $m+p$ of the overall matrix are allowed. Next, the nonzero rows in the $(2,1)$ and $(n,1)$ blocks are eliminated, and a set of r dense rows is introduced into the $(n,2)$ block as a result. In addition, the first p rows of the $(2,2)$ block need to be updated. The elements in the $(1,2)$ block are not altered, except perhaps for some row interchanges.

Factorization of the $(2,2)$ through (n,n) blocks now proceeds similarly. At the $(n-1)$st stage, it is necessary to eliminate $(p+r)$ dense rows in the $(n,n-1)$ block: p rows from the original matrix, plus r rows introduced by the elimination at the previous stage. For the (n,n) block, only column pivoting is used. The overall strategy remains one of *partial* rather than complete pivoting, since at each step we choose a pivot element from *either* the row *or* the column corresponding to the current pivot position, rather than from some submatrix of the unfactored part of A. The choice between a row and a column search is made according to which will avoid fill-in, as we describe above. Further details on this type of pivoting strategy, and its extension to more general matrices, can be found in [3] and [14].

Ignoring low-order terms, the algorithm takes

$$n[\tfrac{1}{3}m^3+\tfrac{5}{2}m^2p-mp^2+mr(2m-p)] \tag{2.2}$$

flops (where each flop consists of an addition and a multiplication). For $p=r=0$ this reduces to $\frac{1}{3}m^3n$; for $r=0, p=m$ it is $(11/6)m^3n$. The small operation count in the former case arises from the fact that there are no subdiagonal blocks to be eliminated. In the case of fully coupled boundary conditions $(p=0, r=m)$ the flop count is $(7/3)m^3n$, which reflects the work involved in "chasing" elements from the $(n,1)$ block across the bottom of the matrix during factorization.

The SOLVE routine is relatively simple. Given a right-hand side y, the following equations are solved in succession:

$$Lv=Py \quad \text{for } v, \tag{2.3a}$$

$$Uw=v \quad \text{for } w, \tag{2.3b}$$

$$u=Qw \quad \text{for the final answer } u. \tag{2.3c}$$

The forward substitution (2.3a) requires about $nm(p+r)$ flops, while (2.3b) requires about $nm(2m-p)$, making a total of

$$nm(2m+r) \tag{2.4}$$

operations for the whole process. For the case $r=0$ this reduces to $2nm^2$, and hence requires between $12/11m$ and $6/m$ of the execution time of DECOMP (depending on the value of p). For small values of m, therefore, the SOLVE phase is a significant part of the overall computation, particularly if p is also small. This effect becomes even more important when it is necessary to call SOLVE more than once for each call to DECOMP. This happens in PASVA4 for two reasons. First, a modified Newton's method is used to solve (1.2), in which the Jacobian is not necessarily updated at each iteration, obviating a refactorization. Second, PASVA4 allows the differential system (1.1) to be augmented by some algebraic equations and, correspondingly, some extra parameters. The linear system therefore has a "border" of extra rows and columns, and a Schur-complement approach involving one extra call to SOLVE for each extra parameter, is used to solve it. This occurs in the application to be discussed in § 5, in which SOLVE is called up to 10 times more often than DECOMP.

3. The new algorithms. In this section we report on four new algorithms for the factorization of the Jacobian, all based on the equation and variable ordering of Fig. 2. Three new methods that use the resulting L and U factors are also needed for the solution phase. The common feature of the four factorization algorithms is that the LU factorizations of blocks $(1, 1)$ through $(n-1, n-1)$ are formed concurrently. Row partial pivoting is used within each block. The algorithms differ in how the fill-in elements in the last row of blocks are computed, and also in whether, and how, the superdiagonal blocks are altered by the factorization (as they are not, in the original DECOMP).

In all the algorithms, the most significant parts of the computation take place in loop constructs of one of the following forms:

```
(3.1)   General triad:   for $i = 1, 2, \cdots$
                           for $j = 1$ to $t$
                             $a_i(j) \leftarrow a_i(j) + b(j) * c_i(j)$
(3.2)   Inner product:   for $i = 1, 2, \cdots$
                           for $j = 1$ to $t$
                             $\alpha_i \leftarrow \alpha_i + a_i(j) * b(j)$
(3.3)   Saxpy:   for $i = 1, 2, \cdots$
                   for $j = 1$ to $t$
                     $a(j) \leftarrow a(j) + \beta_i * c_i(j)$
(3.4)   Saxpy':  for $i = 1$ to $t$
                   for $j = 1$ to $i$
                     $a(j) \leftarrow a(j) + \beta_i * c_i(j)$
```

Here a_i, b, c_i are vectors whose jth components are $a_i(j)$, $b(j)$, $c_i(j)$.

On a vector processor, the time required for the inner loop in each of the above structures is comprised of

(a) The time to fetch the argument vectors from main (or cache) memory.

(b) The time required to initialize the pipeline.

(c) Processing time for each component of the result (typically, after the start-up time, one component of the result appears at the end of the pipe at each clock cycle).

(d) The time required to store the result.

On register-based machines (including the Alliant and all CRAY machines), this overhead may be incurred repeatedly as the vectors are processed in "strips" of 32 or 64 vector components. For the general triad (3.1), it is necessary to load two vectors for each new value of i (except for three vectors when $i = 1$), and store one result. For the inner product in (3.2), it is usually only necessary to load one vector and store a scalar result for each value of i. In (3.3) and (3.4), there is usually one vector load for each i, and no store until the end.

The operations (a)-(d) above are implemented quite differently on different architectures. For example, on the CRAY X-MP and Y-MP machines, two vectors can be loaded almost simultaneously, while on the CRAY 1 and CRAY 2, the second load cannot start until the first is complete. Preparation of the arithmetic processing pipe takes a different number of clock cycles on different machines. On the Alliant, the flop at each inner iteration of the loops in (3.1)-(3.4) is performed by a single "add-multiply" instruction; on the CRAYs, the add and multiply are done separately (although they are "chained" to give the effect of a single command).

We use T_t, R_t, S_t, and S'_t to denote the average time required for each iteration of the inner loop in (3.1), (3.2), (3.3), and (3.4), respectively. Each of these quantities could be expressed in terms of more fundamental parameters, such as clock-cycle time, main/cache memory fetch time, vector-register length, etc., but the resulting expressions would be nonlinear (and rather complex) functions of t. Although each of these quantities is generally monotonically decreasing with t, there may be sharp increases at certain values of t (e.g., multiples of the vector register length). The relation $S_t < S'_t$ should hold in general.

For the purpose of the algorithm descriptions below, we introduce the following notation for the various blocks:

$$A_i = \text{the } (i, i) \text{ block},$$

$$B_i = \text{the } (i, i+1) \text{ block},$$

$$D = \text{the } (n, 1) \text{ block}.$$

Using this notation, Fig. 2 can be rewritten as

$$A = \begin{pmatrix} A_1 & B_1 & & & \\ & A_2 & B_2 & & \\ & & \ddots & \ddots & \\ & & & & B_{n-1} \\ D & & \cdots & & A_n \end{pmatrix}. \tag{3.5}$$

The first two algorithms produce a true LU factorization, that is,

$$LU = PA, \tag{3.6a}$$

where the L and U factors and the permutation matrix have the form

$$L = \begin{pmatrix} L_1 & & & \\ & L_2 & & \\ & & \ddots & \\ E_1 & \cdots & E_{n-1} & L_n \end{pmatrix}, \quad U = \begin{pmatrix} U_1 & V_1 & & & \\ & U_2 & V_2 & & \\ & & \ddots & \ddots & \cdot \\ & & & & V_{n-1} \\ & & & & U_n \end{pmatrix}, \tag{3.6b}$$

$$P = \begin{pmatrix} P_1 & & & \\ & P_2 & & \\ & & \ddots & \\ & & & P_n \end{pmatrix}.$$

Here the L_i are unit lower triangular, U_i are upper triangular, and P_i are $m \times m$ permutation matrices. The matrices $E_1, \cdots, E_{n-1}$ each have $p+r$ dense rows, the positions of which depend on the permutation matrix P_n. Relating (3.5) and (3.6), we find that equations for the submatrices are:

$$L_iU_i = P_iA_i, \qquad i = 1, \cdots, n-1, \tag{3.7}$$

$$L_iV_i = P_iB_i, \qquad i = 1, \cdots, n-1, \tag{3.8}$$

$$E_1U_1 = P_nD, \tag{3.9a}$$

$$E_iV_i + E_{i+1}U_{i+1} = 0, \qquad i = 1, \cdots, n-2, \tag{3.9b}$$

$$E_{n-1}V_{n-1} + L_nU_n = P_nA_n. \tag{3.9c}$$

If the matrix A in (3.5) is partitioned as

$$A = \begin{pmatrix} & & & & \vdots & 0 \\ & C & & & \vdots & \vdots \\ & & & & \vdots & B_{n-1} \\ & & & & \vdots & \\ \cdots & \cdots & \cdots & \cdots & & \cdots \\ D & 0 & \cdots & 0 & \vdots & A_n \end{pmatrix}$$

it can be seen that the factorization (3.6) is obtained by finding an LU factorization of C with (unrestricted) partial pivoting, then computing

$$[D \quad 0 \cdots 0]C^{-1} = [E_1 \quad E_2 \quad \cdots \quad E_{n-1}],$$

and finally computing an LU factorization of the Schur complement of C in A, namely,

$$A_n - \{[D \quad 0 \cdots 0]C^{-1}\}\begin{pmatrix} 0 \\ \vdots \\ 0 \\ B_{n-1} \end{pmatrix}.$$

All our algorithms are based on the calculation of

$$[D \quad 0 \cdots 0]C^{-1}$$

rather than

$$C^{-1}\begin{pmatrix} 0 \\ \vdots \\ 0 \\ B_{n-1} \end{pmatrix}.$$

To obtain the latter, the solution of m linear systems with coefficient matrix C are required, while for the former, only $p+r$ linear systems with coefficient matrix C^T need to be solved.

The first algorithm follows directly from (3.7)-(3.9).

Algorithm D1.

```
(1) for $i = 1$ to $n-1$
        factor $L_iU_i = P_iA_i$
(2) for $i = 1$ to $n-1$
        solve $L_iV_i = P_iB_i$ for $V_i$
(3) solve $\bar{E}_1U_1 = D$ for $\bar{E}_1$
```

```
(4) for $i=1$ to $n-2$
        solve $\bar{E}_i V_i + \bar{E}_{i+1} U_{i+1} = 0$ for $\bar{E}_{i+1}$
(5) factor $L_n U_n = P_n(A_n - \bar{E}_{n-1} V_{n-1})$
(6) set $E_i = P_n \bar{E}_i$, $i = 1, \cdots, n-1$.
```

The computationally significant steps in this algorithm are steps (1), (2), and (4). Step (1) requires about $\frac{1}{3}nm^3$ flops, step (2), about $\frac{1}{2}nm^3$, and step (4) about $\frac{3}{2}nm^2(p+r)$ flops. However, the factorizations in step (1) and backsolves in step (2) can be carried out concurrently, that is, they can be coded in the form of (3.1), where the innermost loop has $n-1$ iterations. The iterations in step (4), on the other hand, must be carried out in sequence. The only possibilities for vectorization in this step involve vectors of length m or less. The formation of $\bar{E}_i V_i$ can be coded in the form (3.3). The back substitution for $\bar{E}_{i+1}$ can be done in two ways: it can use either the structure (3.3), with $t=p+r$, or the structure (3.4), with $t=m$. Which of these is faster obviously depends on the value of $p+r$. Since the "average" length of the loop in (3.4) is $t/2$ (i.e., $m/2$ in our case), a simple switch in the code causes it to use (3.3) when $p+r>m/2$, and (3.4) otherwise. The approximate time required for Algorithm D1 will thus be

$$\frac{5}{6}nm^3 T_{n-1} + nm^2(p+r)S_m + \frac{1}{2}nm^2(p+r)\min(S'_m, S_{p+r}). \tag{3.10}$$

We note in passing that in the decoupled case ($r=0$), we can make the assumption that $p \leqq m/2$. If this is not true, we can simply reverse the order of the unknowns and rearrange the equations appropriately so that p and q are interchanged. This corresponds to reversing the sign of the independent variable t in (1.1).

The second algorithm is a variant of Algorithm D1, in which step (4) is partially vectorized at the expense of an increased operation count. From (3.7)-(3.9) and step (6) of Algorithm D1 we have by a simple inductive argument that

$$\bar{E}_i V_i = (-1)^{i+1} D \prod_{j=1}^{i} W_j, \quad \text{where } W_j = U_j^{-1} V_j. \tag{3.11}$$

Given that $(\bar{E}_i V_i)$ is known for $i=1, \cdots, n-2$, it is possible to use forward substitution to calculate the matrices $\bar{E}_{i+1}$ concurrently. The algorithm can be summarized as follows.

Algorithm D2.

```
(1) for $i=1$ to $n-1$
        factor $L_i U_i = P_i A_i$
(2) for $i=1$ to $n-1$
        solve $L_i V_i = P_i B_i$ for $V_i$
(3) for $i=1$ to $n-1$
        solve $U_i W_i = V_i$ for $W_i$
(4) set $\bar{Z}_1 = D$
        for $i=1$ to $n-1$
            set $\bar{Z}_{i+1} = -\bar{Z}_i W_i$
(5) for $i=1$ to $n-1$
        solve $\bar{E}_i U_i = \bar{Z}_i$ for $\bar{E}_i$
(6) factor $L_n U_n = P_n(A_n + \bar{Z}_n)$
(7) set $E_i = P_n \bar{E}_i$, $i = 1, \cdots, n-1$.
```

The new steps in Algorithm D2 are (3), (4), and (5). Steps (3) and (5) are "vectorizable" (i.e., contain operations involving vectors of length at least $n-1$), with flop counts of about $\frac{1}{2}nm^3$ and $\frac{1}{2}nm^2(p+r)$, respectively. Step (4) is "sequential," but noting that each Z_i has only $(p+r)$ dense rows, we count about $nm^2(p+r)$ flops. The approximate execution time is thus

$$[\tfrac{4}{3}nm^3+\tfrac{1}{2}nm^2(p+r)]T_{n-1}+nm^2(p+r)S_m, \qquad p+r>0, \tag{3.12a}$$

$$\tfrac{5}{6}nm^3T_{n-1}, \qquad p+r=0. \tag{3.12b}$$

(In the case (3.12b) steps (3), (4), and (5) are not necessary.) There is a trade-off in Algorithm D2: the total operation count is increased, but the number of "sequential" operations is decreased.

The solution phase corresponding to Algorithms D1 and D2 is the same, and referring to (3.6), it can be stated as follows.

ALGORITHM S1. Given a right-hand side $y^T=[y_1^T,\cdots,y_n^T]$:

```
(1) for $i=1$ to $n-1$
        solve $L_iv_i=P_iy_i$
(2) replace $y_n$ by $(P_ny_n-\sum_{i=1}^{n-1}E_iv_i)$
(3) solve $L_nv_n=y_n$
        solve $U_nu_n=v_n$
(4) for $i=n-1$ to $1$
        replace $v_i$ by $(v_i-V_iu_{i+1})$
        solve $U_iu_i=v_i$.
```

Here, steps (1) and (2) are vectorizable: step (1) requires about $\frac{1}{2}nm^2$ flops, while step (2) requires about $nm(p+r)$ operations. In fact, step (2) consists of inner products involving vectors of length $(n-1)m$ (if the matrices E_i are stored appropriately). The other computationally significant step is step (4), which is sequential in nature and requires about $\frac{3}{2}nm^2$ operations. The time required for Algorithm S1 is thus

$$nm(p+r)R_{(n-1)m}+\tfrac{1}{2}nm^2T_{n-1}+nm^2S_m+\tfrac{1}{2}nm^2S'_m. \tag{3.13}$$

Again we note that a comparison of (3.13) with (3.10) and (3.12), shows that the solution phase is not insignificant in relation to the factorization phase unless m is relatively large.

In the third algorithm, a block-LU factorization is performed. Matrices $\tilde{L}$, $\tilde{U}$, and P are obtained such that $\tilde{L}\tilde{U}=PA$, where

$$\tilde{L}=\begin{pmatrix} I & & & \\ & \ddots & & \\ & & I & \\ D_1 & \cdots & D_{n-1} & I\end{pmatrix}, \qquad \tilde{U}=\begin{pmatrix} \tilde{A}_1 & \tilde{B}_1 & & & \\ & \tilde{A}_2 & \tilde{B}_2 & & \\ & & \ddots & \ddots & \\ & & & \tilde{A}_{n-1} & \tilde{B}_{n-1} \\ & & & & \tilde{A}_n\end{pmatrix} \tag{3.14}$$

and P is as in (3.6b). In (3.14), the matrices $\tilde{A}_1,\cdots,A_n$ are overwritten in storage by their L and U factors. Comparing (3.14) and (3.5), we obtain the formulae

$$\begin{aligned}
&\tilde{A}_i=P_iA_i, && i=1,\cdots,n-1,\\
&\tilde{B}_i=P_iB_i, && i=1,\cdots,n-1,\\
&D_1\tilde{A}_1=P_nD, &&\\
&D_i\tilde{B}_i+D_{i+1}\tilde{A}_{i+1}=0, && i=1,\cdots,n-2,\\
&D_{n-1}\tilde{B}_{n-1}+\tilde{A}_n=P_nA_n. &&
\end{aligned}$$

These suggest the following algorithm.

ALGORITHM D3.
```
(1) for $i=1$ to $n-1$
        factor $L_iU_i = P_iA_i = \tilde{A}_i$
(2) solve $\bar{D}_1\tilde{A}_1 = D$
(3) for $i=1$ to $n-2$
        solve $\bar{D}_i\tilde{B}_i + \bar{D}_{i+1}\tilde{A}_{i+1} = 0$ for $\bar{D}_{i+1}$
(4) factor $L_nU_n = P_n(A_n - \bar{D}_{n-1}\tilde{B}_{n-1})$
(5) set $D_i = P_n\bar{D}_i$, $i=1,\cdots,n-1$.
```

The only step that can be efficiently vectorized here is step (1), which requires about $\frac{1}{3}nm^3$ operations. Step (3) is strictly sequential, requiring a matrix multiplication $\bar{D}_i\tilde{B}_{i+1}$ and two triangular solves using the L and U factors of A_{i+1}. (The L and U factors of each diagonal block are the same as those computed in Algorithm D1, and Algorithm D2, provided the same pivot strategy is used within each block.) The workload for step (3) is $2nm^2(p+r)$ operations, and the approximate total time for Algorithm D3 will thus be

$$\frac{1}{3}nm^3T_{n-1} + nm^2(p+r)S_m + nm^2(p+r)\min(S'_m, S_{p+r}). \tag{3.15}$$

The solution phase of course differs from Algorithm S1, and can be stated as follows.

ALGORITHM S3. Given a right-hand side $y^T = [y_1^T, \cdots, y_n^T]$:
```
(1) for $i=1$ to $n$
        replace $y_i$ by $P_iy_i$
(2) replace $y_n$ by $(P_ny_n - \sum_{i=1}^{n-1} D_iy_i)$
(3) solve $L_nU_nu_n = y_n$ for $u_n$
(4) for $i=1$ to $n-1$
        solve $L_iU_iu_i = y_i - \tilde{B}_iu_{i+1}$ for $u_i$.
```

Only the second step (inner products with vectors of length $(m(n-1))$) can be efficiently vectorized here. The approximate time required is therefore

$$nm(p+r)R_{(n-1)m} + nm^2S_m + nm^2S'_m. \tag{3.16}$$

A fourth combination of routines, suggested by a referee, is also based on a block factorization. Here we find matrices $\hat{L}$, $\hat{U}$, and P such that $\hat{L}\hat{U} = PA$, where P is as in (3.6) and $\hat{L}$ and $\hat{U}$ have the form

$$\hat{L} = \begin{pmatrix} \tilde{A}_1 & & & & \\ & \tilde{A}_2 & & & \\ & & \ddots & & \\ & & & \tilde{A}_{n-1} & \\ Z_1 & \cdots & & Z_{n-1} & \tilde{A}_n \end{pmatrix}, \quad \hat{U} = \begin{pmatrix} I & W_1 & & & \\ & I & W_2 & & \\ & & \ddots & \ddots & \\ & & & & W_{n-1} \\ & & & & I \end{pmatrix}.$$

Again, a comparison with (3.5) yields the formulae

$$\begin{aligned} &\tilde{A}_i = P_iA_i, \qquad i=1,\cdots,n-1,\\ &\tilde{A}_iW_i = P_iB_i, \qquad i=1,\cdots,n-1,\\ &Z_1 = P_nD,\\ &Z_{i+1} = -Z_iW_i, \qquad i=1,\cdots,n-2,\\ &Z_{n-1}W_{n-1} + \tilde{A}_n = P_nA_n. \end{aligned}$$

Note that the W_i and $\tilde{A}_i$ in the formulae above are the same as those in Algorithm D2, while the Z_i above are identical to $P_n\bar{Z}_i$ from Algorithm D2. The resulting algorithm is likewise very similar to Algorithm D2, the main different being omission of step (5):

```
ALGORITHM D4.
(1) for $i=1$ to $n-1$
        factor $L_iU_i = P_iA_i$
(2) for $i=1$ to $n-1$
        solve $L_iV_i = P_iB_i$ for $V_i$
(3) for $i=1$ to $n-1$
        solve $U_iW_i = V_i$ for $W_i$
(4) set $\bar{Z}_1 = D$
    for $i=1$ to $n-1$
        set $\bar{Z}_{i+1} = -\bar{Z}_iW_i$
(5) factor $L_nU_n = P_n(A_n+\bar{Z}_n)$
(6) set $Z_i = P_n\bar{Z}_i$, $i=1,\cdots,n-1$.
```

The approximate run time is

$$\frac{4}{3}nm^3T_{n-1}+nm^2(p+r)S_m. \tag{3.17}$$

Only steps (4) and (6) can be omitted when $p+r=0$. The corresponding solution phase follows.

```
ALGORITHM S4.
(1) for $i=1$ to $n-1$
        solve $L_iv_i = P_iy_i$
(2) for $i=1$ to $n-1$
        solve $U_iw_i = v_i$
(3) replace $y_n$ by $(P_ny_n - \sum_{i=1}^{n-1} Z_iw_i)$
(4) solve $L_nU_nu_n = y_n$
(5) for $i=n-1$ to $1$
        set $u_i = w_i - W_iu_{i+1}$.
```

Here steps (1), (2), and (3) are all vectorizable. The approximate execution time is thus

$$nm(p+r)R_{(n-1)m}+nm^2T_{n-1}+nm^2S_m. \tag{3.18}$$

When $T_{n-1}<S_m \leqq S_{m'}$, we see that none of the Algorithms D1-D4 is superior to any one of the others in all situations. However when $p+r=0$, Algorithm D3 is clearly fastest, followed by Algorithms D1 and D2 (which are identical in this case) and finally Algorithm D4. For $p+r>0$, it is clear by comparing (3.12a) and (3.17) that Algorithm D4 will be faster than Algorithm D2, while for $p+r=m$, Algorithm D4 will be the fastest, and Algorithm D1 will be faster than Algorithm D3. All other relationships between run times of the factorization algorithms will depend on the relative values of p, r, m and the execution times S'_m, S_{p+r}, and T_{n-1}.

For the solve algorithms, making the same assumption that $T_{n-1}<S_m \leqq S_{m'}$ it is clear that Algorithm S4 is always fastest, followed by Algorithm S1 and then Algorithm S3.

4. Computational comparisons. Here we report on computational experience with the algorithms of the two preceding sections, on three computers with vectorization capabilities. Matrices of the form (3.5) were generated in order to test Algorithms D1-D4, and these same matrices were rearranged into the form of Fig. 1 to test the original DECOMP. In accordance with the purpose of the exercise, the form of these matrices corresponds roughly to what would be obtained from a finite-difference discretization of (1.1), that is,

$$A_i = I + h\bar{A}_i, \quad B_i = -I + h\bar{B}_i, \quad i = 1, \cdots, n-1$$

where the elements of $\bar{A}_i$ and $\bar{B}_i$ are uniformly distributed in $[-1, 1]$, and $h = .1$. The nonzero elements of A_n and D are uniformly distributed in $[-h, h]$.

The new factorization algorithms contain various checks for ill-conditioning. The largest and smallest elements of the U_i blocks are monitored throughout the factorization. In Algorithms D2 and D4, a comparison of the Frobenius norm of the matrices A_n and Z_n is made, to detect possible blowup as a result of the matrix multiplications in step (4).

The three computers used were an Alliant FX/8 vector multiprocessor (at the Advanced Computing Research Facility, Argonne National Laboratory), a CRAY X-MP/48 (at the Pittsburgh Supercomputer Center), and a CRAY-2 (at Cray Research, Mendota Heights, Minnesota).

The Alliant is a machine with eight processors (known as computational elements or CEs), which share a main memory and a 512 Kbyte fast-access cache memory. Each computational element can perform operations on vector registers that hold up to 32 double-precision (8-byte) words. The peak performance of each CE in single precision is 11.7 Mflops. When a data element is referenced by a program, it is automatically read into the cache, so that subsequent accesses to that element take substantially less time. However if the data references in the program are not localized, the element may be overwritten in cache, and will thus have to be fetched from main memory again when next referenced. This occurrence is referred to as a "cache fault." Careful management of the cache memory can lead to sharply reduced runtimes (see, for example, [4]).

Only one processor of each CRAY is used. The Pittsburgh CRAY X-MP has a clock-cycle time of 8.5 ns, and a common memory of 8 Mwords, arranged in 32 banks. To load a word of data from main memory, 14 clock cycles are required; to store a word takes somewhat less time. A set of 64-word vector registers is available, together with a corresponding set of vector instructions. To illustrate the processing capabilities of the X-MP, consider its performance in computing the general triad $d(j) = a(j) + b(j)c(j), j = 1, \cdots, t$ (the same as (3.1), except that three vector loads and one store are required). For $t = 10$ the throughput is 25.2 Mflops, for $t = 100$ it is 73.7 Mflops, and for $t = 1000$, 91.1 Mflops (see Schönauer [13]). By comparison, the peak rates for multiplication and addition in scalar mode are 15 Mflops and 17.5 Mflops, respectively. On the CRAY-2, the cycle time is 4.1 ns, and the main memory consists of 256 Mwords organized into 128 banks. To retrieve a word from main memory, 57 clock cycles are required. However, data can be retrieved from each of the banks simultaneously, and so components of a long vector are typically available for processing at the rate of one per clock cycle after this initial delay (although problems of "section conflict" can arise; see [13]). The long fetch time is also partially offset by the presence of 16 Kwords of local (cache) memory for each processor. Access time for this storage is much shorter—four clock cycles. A set of eight 64-word vector registers can be used by each processor.

Although the expressions for approximate execution times were derived in the previous section with vectorization in mind, the algorithms were also run on the Alliant in "scalar" and "parallel-vector" mode. In scalar mode with global optimization (i.e., the compiler optimizations `-Og -DAS` were used), the program runs on a single CE without using any vectorization capabilities. In parallel-vector mode (i.e., using the compiler options `-O -DAS -alt`), the compiler attempts to vectorize the innermost loop of any nested loop structure, and to schedule different iterations of some outer loop to execute concurrently on different CEs. This arrangement may be disturbed if later iterations of a loop depend on the results of earlier iterations (i.e., data dependencies), or if user-inserted compiler directives are present in the code. For details see [1]. In some of the vectorizable steps of the algorithms in § 3 (e.g., step (2) of Algorithm D1, steps (2) and (3) of Algorithm D2) it is possible to arrange for the "parallelized" loop to have length m, so that the step can take full advantage of the parallelism described above.

Results are given for three problem sets. In the first, the values $m=7$, $r=0$, $n=20$ are used; for the second, $m=7$, $r=0$, $n=100$. (These choices were motivated by the application to be discussed in § 5.) For the final problem set we use $m=32$, $r=0$, $n=100$. In all cases, the values of p and q were varied between 0 and m and timings are given for some of these values. Times were measured using the utility functions `second` (on the CRAYs) and `etime` on the Alliant. Readings taken at different times showed that they appeared to be relatively independent of the system load, and they are almost certainly accurate to within 10 percent.

The codes for all machines were identical, except for the use of different timing routines, and the insertion of different compiler directives. Single precision (24 bits of accuracy on the Alliant, 53 bits of accuracy on the CRAYs) was used throughout. On the CRAY machines, the cft77 compiler was used, with options `-esaq -dp` for the vectorized versions, and `-esaq -dp -o novector` for the nonvectorized versions.

Tables 1-6 give timings obtained for the first data set. For the vectorized versions of the codes on the Alliant (Tables 1 and 2), the results correspond closely to what we might expect from the timing expressions of the previous section. When $p=0$ and three or fewer solves are needed for each factorization, D3/S3 is the preferred

TABLE 1
Alliant. Scalar/vectorized/vector-concurrent times (in ms) *for decomposition routines,* $m=7$, $n=20$.

p	DECOMP	D1	D2	D3	D4
0	28./24./15.	34./22./14.	34./22./13.	20./16./11	52./28./14.
1	47./41./27.	40./25./22.	72./33./18.	30./24./24.	54./28./17.
4	80./75./47.	70./50./24.	97./43./18.	68./58./26.	73./39./18.
7	89./85./51.	97./62./26.	122./55./20.	99./68./29.	91./50./20.

TABLE 2
Alliant. Scalar/vectorized/vector-concurrent times (in ms) *for solution routines,* $m=7$, $n=20$.

p	SOLVE	S1	S3	S4
0	10./7.7/5.6	11./6.9/5.2	9.9/8.2/6.7	12./5.4/3.4
1	10./7.8/6.2	11./7.2/5.5	10./8.3/6.7	12./5.6/3.5
4	9.9/7.9/6.3	12./7.4/5.5	12./8.6/6.9	13./5.9/3.6
7	10./7.8/5.7	13./7.7/5.6	13./8.9/7.1	15./6.1/3.6

TABLE 3
CRAY X-MP. *Scalar/vectorized times* (*in* ms) *for decomposition routines*, $m = 7$, $n = 20$.

p	DECOMP	D1	D2	D3	D4
0	2.4/1.2	3.1/0.9	3.0/1.0	1.7/0.8	5.0/1.3
1	4.1/3.2	4.1/1.8	6.3/1.8	3.1/2.1	5.3/1.6
4	7.0/8.1	6.5/2.9	8.7/2.7	6.2/3.3	6.8/2.6
7	8.3/9.6	8.8/4.0	11./3.6	8.9/4.5	8.4/3.6

TABLE 4
CRAY X-MP. *Scalar/vectorized times* (*in* ms) *for solution routines*, $m = 7$, $n = 20$.

p	SOLVE	S1	S3	S4
0	1.0/1.5	1.0/0.5	1.1/0.6	0.9/0.3
1	1.0/1.5	1.1/0.5	1.2/0.6	0.9/0.3
4	1.0/1.5	1.1/0.5	1.2/0.6	1.0/0.3
7	1.0/1.5	1.2/0.5	1.3/0.6	1.1/0.3

TABLE 5
CRAY-2. *Scalar/vectorized times* (*in* ms) *for decomposition routines*, $m = 7$, $n = 20$.

p	DECOMP	D1	D2	D3	D4
0	2.8/2.0	2.8/1.1	2.8/1.3	1.6/0.9	4.1/1.5
1	5.2/4.3	3.9/2.1	6.0/2.3	3.1/2.5	5.0/1.9
4	9.1/9.2	6.3/3.6	8.4/3.5	6.2/4.0	6.7/3.1
7	10./11.	8.6/4.8	11./4.7	9.1/5.3	8.5/4.2

TABLE 6
CRAY-2. *Scalar/vectorized times* (*in* ms) *for solution routines*, $m = 7$, $n = 20$.

p	SOLVE	S1	S3	S4
0	1.3/1.4	1.3/0.7	1.5/0.9	1.1/0.4
1	1.3/1.4	1.3/0.7	1.5/0.9	1.1/0.5
4	1.3/1.4	1.4/0.8	1.6/0.9	1.2/0.5
7	1.3/1.4	1.5/0.8	1.6/0.9	1.3/0.5

combination. In almost all other cases, D4/S4 will be fastest. The combination D4/S4 is always fastest in parallel-vector mode. Note that the speedups (defined as the ratio of runtime for the vectorized implementation to runtime for the parallel-vector implementation) are not impressive—nowhere greater than three on this 8-processor machine. This is due to the fine-grained nature of the computation for this value of n, and the fact that in step (1) of each factorization algorithm, the number of vector operations that can be scheduled to run in parallel decreases from six to one during the computation. The original combination of DECOMP and SOLVE only remains competitive in scalar mode.

On the CRAY X-MP (Tables 3 and 4), the improvement due to vectorization is seen to best advantage in Algorithms D2 and D4. In some cases, the vectorized implementation of DECOMP and SOLVE is actually slower than the scalar implementation, because the overhead associated with the vector instructions is not worthwhile for vectors with seven elements. Among scalar implementations, there is little to choose between the various possible combinations. For the vectorized implementations, the combination D4/S4 is superior, except possibly when $p=0$.

Times for the scalar CRAY-2 codes (Tables 5 and 6), are similar to those for the scalar CRAY X-MP codes, but the speedups due to vectorization are a little smaller. Possibly this is because, on the CRAY-2, the longer memory cycle time means that there is less difference between $R_{m(n-1)}$, T_{n-1}, and S_m. For scalar implementations, there is little to choose between D1/S2, D4/S4 and D3/S3 except for small p, when the latter is best. For the vectorized codes, D4/S4 is superior.

Results for the second data set appear in Tables 7-12. Most of the comments for the first data set apply again here, although for some algorithms (particularly the most highly vectorized codes D2, D4, and S4) the improvement due to vectorization is more pronounced. That is, T_{99} is substantially smaller than T_{19}. The combination D4/S4 is now best in almost all vectorized and parallel-vector implementations. Among scalar implementations there is not much difference between the various possibilities. On the

TABLE 7
Alliant. Scalar/vectorized/vector-concurrent times (in s) for decomposition routines, $m=7$, $n=100$.

p	DECOMP	D1	D2	D3	D4
0	.14/.11/.065	.14/.063/.052	.14/.064/.051	.072/.048/.071	.23/.085/.055
1	.24/.20/.13	.19/.11/.098	.34/.12/.072	.15/.11/.11	.25/.10/.073
4	.41/.39/.24	.35/.23/.11	.47/.17/.073	.34/.28/.12	.35/.16/.077
7	.46/.43/.26	.48/.29/.11	.59/.22/.076	.49/.33/.12	.44/.21/.080

TABLE 8
Alliant. Scalar/vectorized/vector-concurrent times (in ms) for solution routines, $m=7$, $n=100$.

p	SOLVE	S1	S3	S4
0	50/38/28	53/33/25	51/41/33	59/23/14
1	51/39/31	55/34/26	53/41/33	61/23/14
4	50/39/31	61/35/26	59/43/33	67/24/14
7	50/39/28	67/36/26	64/44/34	73/25/14

TABLE 9
CRAY X-MP. *Scalar/vectorized times* (*in* ms) *for decomposition routines*, $m = 7$, $n = 100$.

p	DECOMP	D1	D2	D3	D4
0	14./6.0	15./3.7	15./4.1	8.1/3.3	23./4.9
1	21./17.	21./8.3	30./6.9	16./11.	26./6.5
4	36./43.	33./14.	42./12.	31./16.	34./11.
7	43./50.	45./19.	54./16.	46./22.	42./16.

TABLE 10
CRAY X-MP. *Scalar/vectorized times* (*in* ms) *for solution routines*, $m = 7$, $n = 100$.

p	SOLVE	S1	S3	S4
0	5.0/7.5	5.3/2.4	5.8/2.8	4.4/1.1
1	5.1/7.6	5.3/2.4	5.9/2.8	4.5/1.1
4	5.1/7.6	5.6/2.5	6.2/2.9	4.9/1.1
7	5.2/7.6	5.9/2.5	6.4/2.9	5.2/1.2

TABLE 11
CRAY-2. *Scalar/vectorized times* (*in* ms) *for decomposition routines*, $m = 7$, $n = 100$.

p	DECOMP	D1	D2	D3	D4
0	14./10.	13./3.8	13./4.5	7.3/3.6	21./5.3
1	27./22.	19./9.0	28./8.3	15./12.	24./6.8
4	47./48.	32./16.	40./14.	31./20.	33./13.
7	53./56.	43./22.	52./20.	46./26.	42./19.

TABLE 12
CRAY-2. *Scalar/vectorized times* (*in* ms) *for solution routines*, $m = 7$, $n = 100$.

p	SOLVE	S1	S3	S4
0	6.6/7.2	6.4/3.4	7.4/4.4	5.1/1.9
1	6.7/7.2	6.6/3.5	7.5/4.4	5.3/1.9
4	6.8/7.3	6.8/3.5	7.8/4.5	5.6/1.9
7	6.7/7.1	7.1/3.6	8.1/4.6	5.9/2.0

Alliant and X-MP, DECOMP/SOLVE is usually best, except when p is small, in which case D3/S3 is competitive.

The results for the third data set (Tables 13-18) have a quite different character. The value of m is now large enough that almost all operations in all the algorithms vectorize efficiently. T_{n-1}, $R_{m(n-1)}$, S_m, and S'_m are now much closer together than for previous data sets, and so the algorithms with the lower operation counts (most notably D1) are more competitive. Overall speedups due to vectorization are quite impressive. Among scalar implementations, the original combination DECOMP/SOLVE is clearly best on all machines. Among vectorized implementations on the CRAY machines, there is little difference between the new algorithms, either for factorization or solve, although the combination D3/S3 has an advantage when $p=0$.

The results on the Alliant (Tables 13 and 14) are distorted by the hierarchical memory structure of that machine. When $m=32$ and $n=100$, the data structures that store the A_i's, the B_i's, and the E_i's occupy about 400 Kbytes each—substantially more than the 256 Kbyte cache can handle. Since data references in the new algorithms tend not to be localized, it is likely that many items of data will cause repeated cache faults during execution of the program. Algorithm D2 (Table 13) is most seriously affected by this phenomenon. For example, the data structure that contains the A_i's (and, after step (1), the L_i and U_i factors), is continually being overwritten in cache, and hence needs to be fetched repeatedly. This occurs at the start of step (3), where the data structure containing the W_i's is initialized to the V_i's. (This is not a problem in Algorithm D4, since the V_i's are not needed in the subsequent solution phase and hence can be overwritten by the W_i's.) It occurs again during step (4), and so the U_i blocks need to be fetched again for step (5). Finally, note that the value $m=32$ allows greater scope for parallelism, and so the speedups in going from vector mode to parallel-vector mode are better for this data set.

We conclude that the new algorithms, in particular Algorithms D4 and S4, can under most circumstances execute significantly faster than DECOMP and SOLVE if implemented in an appropriate way on machines with vectorization capabilities. The

TABLE 13
Alliant. Scalar/vectorized/vector-concurrent times (in s) for decomposition routines, $m=32$, $n=100$.

p	DECOMP	D1	D2	D3	D4
0	5.0/2.2/.60	12./3.4/.98	12./3.6/1.0	4.8/1.7/.62	19./5.5/1.3
2	7.5/3.6/1.3	24./3.9/1.3	27./17./3.2	6.8/2.4/1.0	20./5.8/1.4
16	18./8.2/3.8	24./7.3/1.7	37./24./3.8	20./6.7/1.6	27./7.5/1.7
32	23./12./6.0	36./10./2.2	49./32./4.9	34./10./2.2	34./9.6/2.0

TABLE 14
Alliant. Scalar/vectorized/vector-concurrent times (in s) for solution routines, $m=32$, $n=100$.

p	SOLVE	S1	S3	S4
0	.67/.41/.15	.84/.28/.20	.72/.37/.25	1.1/.28/.15
2	.67/.44/.16	.86/.29/.21	.75/.38/.25	1.1/.29/.15
16	.67/.44/.16	.99/.32/.21	.88/.41/.26	1.2/.32/.16
32	.67/.41/.16	1.1/.36/.22	1.0/.44/.27	1.4/.36/.16

TABLE 15
CRAY X-MP. *Scalar/vectorized times (in* s) *for decomposition routines,* $m = 32$, $n = 100$.

p	DECOMP	D1	D2	D3	D4
0	.59/.089	1.2/.12	1.2/.12	.49/.073	2.0/.18
2	.77/.18	1.4/.15	2.2/.19	.70/.17	2.1/.19
16	1.6/.76	2.3/.28	3.3/.30	1.9/.31	2.7/.29
32	2.1/1.1	3.4/.41	4.6/.43	3.3/.47	3.3/.40

TABLE 16
CRAY X-MP. *Scalar/vectorized times (in* ms) *for solution routines,* $m = 32$, $n = 100$.

p	SOLVE	S1	S3	S4
0	55/36	82/13	79/12	80/11
2	55/35	80/13	79/12	81/12
16	55/36	86/14	86/13	87/12
32	55/36	93/15	93/14	94/13

TABLE 17
CRAY-2. *Scalar/vectorized times (in* s) *for decomposition routines,* $m = 32$, $n = 100$.

p	DECOMP	D1	D2	D3	D4
0	.54/.15	1.1/.14	1.2/.17	.47/.081	1.8/.21
2	.82/.27	1.2/.19	2.1/.27	.67/.21	1.9/.24
16	2.0/.88	2.2/.38	3.1/.43	1.9/.40	2.5/.38
32	2.7/1.2	3.2/.55	4.3/.59	3.2/.59	3.2/.52

TABLE 18
CRAY-2. *Scalar/vectorized times (in* ms) *for solution.*

p	SOLVE	S1	S3	S4
0	83/39	101/24	99/20	91/27
2	83/39	102/25	100/21	92/27
16	83/38	108/25	106/21	97/28
32	83/38	115/27	113/22	104/29

relative efficiency of the various codes varies with the values of *m*, *n*, the distribution of the initial, coupled, and final boundary conditions, and the characteristics of the computers on which they are implemented. However, as seen above, the timing expressions derived in § 3 provide a useful way of predicting the performance.

5. Application to seismic ray tracing. The two-point boundary value problem code PASVA4 has been utilized as part of a program for tracing seismic rays through heterogeneous three-dimensional media by Pereyra (see [11], [12]). The program takes a seismic "event" (earthquake, explosion) at a given location, and traces rays from this source to a network of receivers (e.g., an array of geophones on the earth's surface). The wave propagation properties of the earth in the vicinity of the source and receivers are modeled by a set of blocks limited by surface patches. Parameters that define the interfaces between the blocks, and the properties *within* each block (assumed to be smoothly varying) are given. Initial estimates of the ray paths are obtained by shooting rays from the source in a given set of directions. These paths are refined using PASVA4, which attempts to trace all rays between a given source-receiver pair with a given "signature." The signature is a list of interfaces, which the ray is assumed to contact in a specified order.

PASVA4 is able to handle discontinuities in the material properties due to layer interfaces, and to efficiently consider any additional algebraic parameters present in the problem. Discontinuities are handled by forcing the interface intersections to be repeated meshpoints. Extra conditions (continuity of the ray, laws of refraction) are introduced to determine the additional unknowns. We mentioned in § 2 that the resulting differential-algebraic system can be solved by calling SOLVE more often than DECOMP on each iteration. This is discussed in more detail in [10].

In order to test the effect of the new linear solvers on the overall speed of the ray tracing program, the code was modified so that Algorithm D4 took the place of DECOMP and Algorithm S4 took the place of SOLVE. The original DECOMP was retained as a backup, because of its possibly better stability properties. An "interface" routine was introduced to rearrange the rows of the Jacobian and produce the structure of Fig. 2. This introduces a small overhead, which could be avoided by making appropriate changes to the routines that fill in this matrix.

Comparative results for four problems are given in Table 19. In all problems, rays were traced from a point on the surface to a rectangular network of receivers, also on the surface. Rays with a number of different signatures were obtained. The simplest rays were those which traveled from the source to the first interface, and then were reflected back to a receiver. Other rays penetrated as far as eight interfaces below the surface. Many ray paths were quite complex, because the three-dimensional geometries modeled by the four examples were nontrivial. For example, the problem "fold" contains an interface that literally folds back on itself, and so rays can pass through

TABLE 19
Runtimes in seconds on CRAY-2 *for raytracing codes on four problems.*

	Original code	Modified code
fold	269	175
reverse fault	208	140
norf	154	105
mushroom	1124	849

the fold before or after reflecting from the interface. Similar ray paths can be found in "mushroom," whose main feature is a subsurface salt dome. The original and modified codes produced identical results.

Almost all the runtime for both codes is spent in either executing DECOMP and SOLVE, or in evaluating functions and derivatives associated with the discretized problem. The latter part of the program is not helped much by vectorization. It consists of millions of calls to small routines that perform such tasks as evaluating the splines that define the layer interfaces, or evaluating derivatives with respect to the spline parameters of the point at which a ray contacts the interface. On the other hand, since the values $m = 7$, $p = 4$ are always applicable for ray-tracing problems, and since $n \geqq 18$, we would expect from Tables 1-12 that substantial improvements are possible. This is indeed the case, as we see from the run profiles in Tables 20 and 21. Here, the fold and reverse fault problems are profiled using the flowtrace utility on the CRAY-2. Runtimes are given in absolute terms, and also as a proportion of the runtime for their program. (A slight discrepancy can be detected between these times and those in the relevant lines of Table 19; this is because of the overhead introduced by the flowtrace utility.) In absolute terms, the time needed by DECOMP/SOLVE is reduced by a factor of three, and as a proportion of the total runtime it drops from 45.8 percent to 20.9 percent in the case of the fold problem and 44.7 percent to 22.0 percent in the case of the reverse fault. Note that the number of DECOMP and SOLVE calls is the same for both codes—no problems of stability arose in the use of D4.

Appendix. Validity of the new factorization algorithms. Suppose the matrix considered in § 3 is the Jacobian of a system of nonlinear equations, which arises from a one-step global discretization scheme applied to the problem (1.1)-(1.2). Here we show that if this Jacobian is evaluated in the vicinity of an isolated solution of the nonlinear system, then under mild conditions, the factorization schemes described in § 3 will work, for a sufficiently fine mesh. This is done by referring to some of the results of Keller [8], in particular, those results that prove that the factorization scheme of § 2 is (asymptotically) valid.

TABLE 20
Run profile for fold problem.

	Original code	Modified code
calls to decomp	3776	3776
time in decomp	60.2 secs (21.4%)	18.5 secs (9.3%)
calls to solve	29681	29681
time in solve	68.7 secs (24.4%)	23.1 secs (11.6%)
time in interface	—	1.5 secs (0.8%)

TABLE 21
Run profile for reverse fault problem.

	Original code	Modified code
calls to decomp	3559	3559
time in decomp	44.2 secs (19.2%)	14.3 secs (9.0%)
calls to solve	31772	31772
time in solve	58.7 secs (25.5%)	20.6 secs (13.0%)
time in interface	—	1.4 secs (0.9%)

For simplicity we start by considering a linear version of (1.1):

$$\text{(A1)} \qquad \begin{aligned} &y'(t)-A(t)y(t)=f(t), \qquad a \leqq t \leqq b, \\ &B_a y(a)+B_b y(b)=\beta, \end{aligned}$$

where $f, y, \beta \in R^m$, B_a, B_b, $A \in R^{m\times m}$. In one-step discretization schemes for (A1), the submatrices A_i and B_i in (3.5) have the form

$$\text{(A2)} \qquad \begin{aligned} &A_i(h)=-h_{i+1}^{-1}I+\tilde{A}_i(h), \\ &B_i(h)=h_{i+1}^{-1}I+\tilde{B}_i(h), \qquad i=1,\cdots,n-1, \end{aligned}$$

where the mesh

$$a=t_1<t_2<\cdots<t_n=b$$

is used, with

$$h_i=t_i-t_{i-1}, \qquad h=\max_i h_i.$$

Here $\|\tilde{A}_i(h)\|$ and $\|\tilde{B}_i(h)\|$ are assumed to be uniformly bounded as functions of h. The boundary condition matrices in (A1) are used to define the last row of blocks in (3.5):

$$\text{(A3)} \qquad D=B_a, \qquad A_n=B_b.$$

The approximate discretized solution of (A1) is then found by solving the following system of linear equations:

$$\text{(A4)} \qquad A^h u^h = F^h,$$

where A^h is the matrix from (3.5) and F^h has the form

$$\text{(A5)} \qquad F^h = \begin{pmatrix} F_1(f,h) \\ \vdots \\ F_{n-1}(f,h) \\ \beta \end{pmatrix}.$$

Each F_i is the right-hand side of the ith equation in the one-step discretization scheme, and

$$u^h = \begin{pmatrix} u_1 \\ \vdots \\ u_n \end{pmatrix}$$

contains the approximate discrete solution of (A1) at each meshpoint.

The following result is a direct consequence of Keller [8, Cor. 2.11] and Isaacson and Keller [6, p. 396].

LEMMA A1. *If the one-step scheme* (A2)-(A5) *is consistent for the initial value problem obtained by setting $B_a = I$ and $B_b = 0$ in* (A1), *then it is stable and consistent for $\underline{all}$ problems* (A1) *that have a unique solution.*

By considering the special case of a result of Keller [8, Thm. 2.20], it can be shown that a factorization similar to that produced by Algorithm D3 is valid.

THEOREM A2. *Let* (A1) *have a unique solution, and let the one-step scheme* (A2)-(A3) *be consistent for the initial value problem form of* (A1) (*with $B_a = I$, $B_b = 0$*).

Then there is a small positive number $\bar{h}$ such that for all $h \leqq \bar{h}$, the factorization

$$A^h = \begin{pmatrix} I & & & \\ & \ddots & & \\ & & I & \\ G_1 & \cdots & G_{n-1} & I \end{pmatrix} \begin{pmatrix} A_1 & B_1 & & \\ & \ddots & \ddots & \\ & & A_{n-1} & B_{n-1} \\ & & & \hat{A}_n \end{pmatrix} \tag{A6}$$

is valid, with no need for pivoting.

Proof. Set $p=0$ in Theorem 2.20 of [8], and note that in this case the partial pivoting strategy defined in Theorem 2.17 of [8] (which is equivalent to the row partial pivoting in DECOMP) is not needed.

The factorization (A6) is simply related to (3.14) by appropriate introduction of permutation matrices; in particular, for the bottom row of blocks,

$$D_i = P_n G_i P_i^T, \qquad 1, \cdots, n-1,$$
$$\tilde{A}_n = P_n \hat{A}_n.$$

The validity of the other factorizations (of Algorithms D1, D2, and D4) now follows directly from the validity of (3.14), since the pivot sequence is the same in both cases. Using the submatrices L_i from (3.6) we can define

$$\hat{L} = \begin{pmatrix} L_1 & & & \\ & L_2 & & \\ & & \ddots & \\ & & & L_n \end{pmatrix}.$$

Then (3.6) and (3.14) can be related by

$$PA^h = LU = (L\hat{L}^{-1})(\hat{L}U) = \tilde{L}\tilde{U}.$$

It remains to extend these results for the linear system (A1) to the nonlinear case (1.1). This is done in [8, § 2.3]. The matrices $\tilde{A}_i(h)$, $\tilde{B}_i(h)$, B_a, and B_b now arise from derivatives of the functions f and g in (1.1) and (1.2). The partly separated nature of the boundary conditions in (1.2) means that the first p rows of B_b and the last q rows of B_a are always zero. The system (A4) is now solved iteratively, with the sequence of vectors u^h now (hopefully) converging to the discrete approximation of the true solution. Basically, the analysis of [8] can be used to show that if the solution of (1.1)-(1.2) is isolated, and the current iterate u^h is sufficiently close to the discrete solution, then the factorization schemes of § 3 are valid.

Simple examples of discretizations for which the various factorization schemes described here may fail on a coarse mesh are provided by the following problem:

$$y'(t) = -2y(t), \qquad t \in [0, T]. \tag{A7}$$

Here $T > 1$ is an integer. The trapezoidal rule discretization (1.2a) on the usual mesh yields the equations

$$y_{i+1}(1+h_{i+1}) + y_i(-1+h_{i+1}) = 0, \qquad i = 1, \cdots, n-1.$$

Applying the boundary condition $y(0) = A$ (for which the exact solution is $y(t) = A\,e^{-2t}$), setting $h_i \equiv 1$, and using the ordering of Fig. 1, we obtain the following coefficient matrix:

$$\begin{pmatrix} 1 & & & & \\ & 2 & & & \\ & & 2 & & \\ & & & \ddots & \\ & & & & 2 \end{pmatrix}.$$

This trivial matrix could clearly be factorized by any "reasonable" algorithm. However, the ordering of Fig. 2 produces

$$\begin{pmatrix} 0 & 2 & & & \\ & 0 & 2 & & \\ & & \ddots & \ddots & \\ & & & 0 & 2 \\ 1 & 0 & \cdots & 0 & 0 \end{pmatrix},$$

and so D1–D4 would fail, since in the notation (3.5), $A_i \equiv 0$. If instead we use the coupled boundary condition

$$y(0) = y(T) = A$$

(for which the exact solution is $y(t) = A\,e^{-2t}/(1+e^{-2T})$), Fig. 1 and Fig. 2 orderings are identical:

$$\begin{pmatrix} 0 & 2 & & & \\ & 0 & 2 & & \\ & & \ddots & \ddots & \\ & & & 0 & 2 \\ 1 & 0 & \cdots & 0 & 1 \end{pmatrix}.$$

All the factorization algorithms (including DECOMP) will fail on this matrix.

Acknowledgments. We thank the Advanced Computing Research Facility at Argonne National Laboratory and the Computer Science Department at Stanford University for the use of their Alliant FX/8 computers. We also thank the Office of Advanced Scientific Computing at the NSF for giving us access to the CRAY X-MP/48 at the San Diego Supercomputer Center, and Cray Research for the use of a CRAY-2 at their Mendota Heights, Minnesota location. Jack Dongarra provided helpful comments regarding the CRAY and Alliant architectures, and we also acknowledge with gratitude the important contributions of the editor and referees of earlier versions of this paper, in particular the referee who suggested the algorithms D4 and S4.

REFERENCES

[1] *Alliant Users Guide*, Alliant Computer Corporation, 1985.

[2] J. R. CASH, *Numerical integration of nonlinear two-point boundary value problems using iterated deferred corrections. I: A survey and comparison of some one-step formulae*, Comput. Maths. Appl., 12A (1986), pp. 1029–1048.

[3] J. C. DIAZ, A. FAIRWEATHER, AND P. KEAST, FORTRAN *packages for solving certain almost block diagonal linear systems by modified alternate row and column elimination*, ACM Trans. Math. Software, 9 (1983), pp. 358–375.

[4] K. GALLIVAN, W. JALBY, U. MEIER, AND A. SAMEH, *The impact of hierarchical memory systems on linear algebra algorithm design*, CSRD Tech. Report, University of Illinois, Urbana, Illinois, October 1986.

[5] M. GOLDMANN, *Vectorization of the multiple shooting method for the nonlinear boundary value problem in ordinary differential equations*, Parallel Comput., 7 (1988), pp. 97–110.

[6] E. ISAACSON AND H. B. KELLER, *Analysis of Numerical Methods*, John Wiley, New York, 1966.

[7] H. B. KELLER, *Accurate difference methods for non-linear two-point boundary value problems*, SIAM J. Numer. Anal., 11 (1974), pp. 305–320.

[8] ———, *Numerical Solution of Two-Point Boundary Value Problems*, CBMS-NSF Regional Conference Series in Applied Mathematics 24, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1976.

[9] M. LENTINI AND V. PEREYRA, *An adaptive finite difference solver for nonlinear two-point boundary value problems with mild boundary layers*, SIAM J. Numer. Anal., 14 (1977), pp. 91–111.

[10] ———, PASVA4: *An* ODE *boundary solver for problems with discontinuous interfaces and algebraic parameters*, Mat. Apl. Comput., (1983), pp. 103–118.

[11] V. PEREYRA, *Improved automatic two-point ray tracing in inhomogeneous three-dimensional media*, Report 87-01, Weidlinger Associates Inversion Project, Weidlinger Associates, Los Altos, CA, 1987.

[12] ———, *Numerical methods for inverse problems in three-dimensional geophysical modeling*, Appl. Numer. Math., 4 (1988), pp. 97–139.

[13] W. SCHÖNAUER, *Scientific Computing on Vector Computers*, North-Holland, Amsterdam, 1987.

[14] J. M. VARAH, *Alternate row and column elimination for solving certain linear systems*, SIAM J. Numer. Anal., 13 (1976), pp. 71–75.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 3, pp. 450–481, May 1990

HYBRID KRYLOV METHODS FOR NONLINEAR SYSTEMS OF EQUATIONS*

PETER N. BROWN† AND YOUCEF SAAD‡

Abstract. Several implementations of Newton-like iteration schemes based on Krylov subspace projection methods for solving nonlinear equations are considered. The simplest such class of methods is Newton's algorithm in which a (linear) Krylov method is used to solve the Jacobian system approximately. A method in this class is referred to as a Newton–Krylov algorithm. To improve the global convergence properties of these basic algorithms, hybrid methods based on Powell's dogleg strategy are proposed, as well as linesearch backtracking procedures. The main advantage of the class of methods considered in this paper is that the Jacobian matrix is never needed explicitly.

Key words. nonlinear systems, Krylov methods, inexact Newton methods, conjugate gradient techniques

AMS(MOS) subject classification. 65H10

1. Introduction.

We consider here several implementations of Newton-like iteration schemes for solving nonlinear systems of equations that we will refer to as *nonlinear Krylov subspace projection methods.* All these methods are based upon the idea of using a basic Newton iteration in which the Newton equations are solved approximately by an available Krylov method. The particular Krylov methods we will consider are *Arnoldi's Method* [22], and the *Generalized Minimum Residual Method* (GMRES) [24]. The Krylov methods have the virtue of requiring almost no matrix storage, resulting in a distinct advantage over direct methods for solving the Newton equations.

To be more specific, consider the nonlinear system of equations

$$F(u) = 0, \tag{1.1}$$

where F is a nonlinear function from $\mathbf{R}^N$ to $\mathbf{R}^N$. Newton's method applied to (1.1) results in the following iteration:

1. Set u_0 = an initial guess.
2. For $n = 0, 1, 2, \cdots$ until convergence do:

$$\begin{aligned} &\text{Solve } J(u_n)\delta_n = -F(u_n), \\ &\text{Set } u_{n+1} = u_n + \delta_n, \end{aligned} \tag{1.2}$$

where $J(u_n) = F'(u_n)$ is the system Jacobian. For large problems, iterative methods are frequently used to solve (1.2) only approximately, giving rise to methods that

* Received by the editors November 20, 1987; accepted for publication (in revised form) March 23, 1989. This work was performed under the auspices of the U.S Department of Energy by the Lawrence Livermore National Laboratory under contract W-7405-Eng-48, and supported by the Department of Energy Office of Energy Research, Applied Mathematical Sciences Research Program.

† Computing and Mathematics Research Division, L-316, Lawrence Livermore National Laboratory, Livermore, California 94550. The work of this author was supported in part by National Science Foundation grant DMS-8506651.

‡ Research Institute for Advanced Computer Science, MS230-5, NASA Ames Research Center, Moffett Field, California 94035. This work was performed while this author was at the Center for Supercomputer Research and Development, University of Illinois, Urbana, Illinois 61801. The work of this author was supported in part by National Science Foundation grants DCR84-10110 and DCR85-09970, by U.S. Department of Energy grant DE-FG02-85ER25001, by U.S. Air Force contract AFSOR-85-0211, and from International Business Machines.

can be viewed as *inexact-Newton* methods [6]. We will refer to a Newton iteration in which a Krylov method is used to solve (1.2) approximately as a *nonlinear Krylov method.*

Typically, a Krylov method for solving (1.2) requires only the action of the Jacobian matrix J times a vector v, and not J explicitly. In the nonlinear equations setting, this action can be approximated by a difference quotient of the form

$$J(u)v \approx \frac{F(u+\sigma v) - F(u)}{\sigma}, \tag{1.3}$$

where u is the current approximation to a root of (1.1) and σ is a scalar. In [2], Brown has given an analysis of the resulting Newton/Krylov algorithms when (1.3) is used to approximate Jv, and has referred to them as *inexact-Newton/finite-difference projection methods.* Sufficient conditions are given in [2] on the size of the σ's in the finite-difference algorithms that guarantee the local convergence of the iteration. Here, we will be concerned with modifications of the above algorithms that are intended to guarantee the global convergence of the iteration to a local solution of (1.1). We recall that Newton's method converges only when the initial guess is close enough to a solution, so a modification is needed to guarantee convergence for arbitrary initial guesses. We refer to a method that converges for any initial guess as *globally convergent*, as opposed to a locally convergent method such as the unmodified Newton iteration. The first modification will add a linesearch backtrack procedure to the basic algorithm, whereas the second will incorporate a local quadratic model of the function $f(u) = \frac{1}{2}F(u)^T F(u)$ and will be a model trust region type algorithm.

We note that several authors have considered Krylov methods for solving the Newton equations approximately inside a Newton algorithm in the context of systems of ordinary differential equations [3]–[5],[11]. Also, Steihaug [25] and O'Leary [20] have used the *Conjugate Gradient* method in the unconstrained optimization of a real-valued function of several variables. Wigton, Yu, and Young [27] and more recently Kerkhoven and Saad [16] have accelerated nonlinear fixed point iterations of the form $u_{n+1} = M(u_n)$ by applying this approach to solving the nonlinear system of equations $u - M(u) = 0$. Note that, as was observed by Chan and Jackson [5], the new system of equations $u - M(u) = 0$ can be viewed as a nonlinearly preconditioned version of the original system of equations. This constitutes one way of preconditioning a nonlinear system and we should stress that it retains the nice feature of not requiring explicit Jacobians. A second approach for preconditioning a nonlinear system of equations is to exploit the fact that often the system separates naturally into a simple linear part, e.g., the Laplacian operator in partial differential equations, and the nonlinear part. In many cases the linear part may constitute a good preconditioner to the whole system, and fast direct solvers can be exploited to apply these preconditioners (e.g., see [1] and [26]). A more standard way of preconditioning a nonlinear system is to use the usual incomplete factorization techniques such as the incomplete *LU* (*ILU*) factorization. However, this approach requires the Jacobian matrix explicitly and thus loses the main advantage of Jacobian-free Krylov subspace methods. Nevertheless, we can easily imagine a procedure where the Jacobian is computed only occasionally, i.e., much less frequently than with a standard Newton approach, in order to derive a preconditioning.

In §2, we review the basic Krylov methods under consideration, and then in §3 we present the linesearch backtracking modification of the nonlinear Krylov iteration. In §4, we present a model trust region algorithm in connection with the nonlinear GMRES method, in §5 we discuss scaling and preconditioning of the linear and nonlinear iterations, and then in §6 we present some numerical results on the above algorithms. Finally, in §7 we make some concluding remarks.

2. Nonlinear Krylov algorithms. In this section we will review the Krylov subspace methods under consideration, and discuss their main properties. We start with a brief description of the Arnoldi and GMRES algorithms, and then present their nonlinear versions, which combine them with a Newton iteration. Next, we comment on some implementation details of Arnoldi and GMRES, and then briefly discuss finite-difference versions and incomplete versions of the two methods.

Once again, we are interested in using a Newton-like iteration scheme to solve the nonlinear system

$$F(u) = 0, \tag{2.1}$$

where F is a nonlinear function from $\mathbf{R}^N$ to $\mathbf{R}^N$. As discussed in the Introduction, at each iteration we must obtain an approximate solution of the linear system (1.2), which we rewrite as

$$J\delta = -F, \tag{2.2}$$

where F and its Jacobian J are evaluated at the current iterate. If $\delta^{(0)}$ is an initial guess for the true solution of (2.2), then letting $\delta = \delta^{(0)} + z$, we have the equivalent system

$$Jz = r^{(0)}, \tag{2.3}$$

where $r^{(0)} = -F - J\delta^{(0)}$ is the initial residual. Let K_m be the *Krylov subspace*

$$K_m \equiv \operatorname{span}\{r^{(0)}, Jr^{(0)}, \cdots, J^{m-1}r^{(0)}\}.$$

Arnoldi's method and GMRES both find an approximate solution

$$\delta^{(m)} = \delta^{(0)} + z^{(m)} \text{ ,with } z^{(m)} \in K_m, \tag{2.4}$$

such that either

$$(-F - J\delta^{(m)}) \perp K_m \text{ (equivalently } (r^{(0)} - Jz^{(m)}) \perp K_m) \tag{2.5}$$

for Arnoldi's method, or

$$\|F + J\delta^{(m)}\|_2 = \min_{\delta \in \delta^{(0)} + K_m} \|F + J\delta\|_2 \ \ (= \min_{z \in K_m} \|r^{(0)} - Jz\|_2) \tag{2.6}$$

for GMRES. Here, $\|\cdot\|_2$ denotes the Euclidean norm on $\mathbf{R}^N$ and orthogonality is meant in the usual Euclidean sense.

The following algorithm is a nonlinear version of the Arnoldi (GMRES) algorithm, which at every outer iteration generates an orthonormal system of vectors v_i $(i = 1, 2, \cdots, m)$ of the subspace K_m and then builds the vector $\delta^{(m)}$ that satisfies (2.5) (or (2.6) for GMRES). In both algorithms, v_1 is obtained by normalizing $r^{(0)}$.

Algorithm: Newton–Arnoldi (Newton–GMRES)

```
(1) *Start:* Choose $u_0$ and compute $F(u_0)$. Set $n = 0$. Choose a tolerance $\epsilon_0$.
(2) *Arnoldi process:*
```

- For an initial guess $\delta^{(0)}$, form $r^{(0)} = -F - J\delta^{(0)}$, where $F = F(u_n)$ and $J = J(u_n)$.
- Compute $\beta = \|r^{(0)}\|_2$ and $v_1 = r^{(0)}/\beta$.
- For $j = 1, 2, \cdots$, do:
  (a) Form Jv_j and orthogonalize it against the previous $v_1, \cdots, v_j$ via

$$\begin{aligned} h_{i,j} &= (Jv_j, v_i), \quad i = 1, 2, \cdots, j, \\ \hat{v}_{j+1} &= Jv_j - \sum_{i=1}^{j} h_{i,j} v_i \\ h_{j+1,j} &= \|\hat{v}_{j+1}\|_2, \quad \text{and} \\ v_{j+1} &= \hat{v}_{j+1}/h_{j+1,j}. \end{aligned} \tag{2.7}$$

  (b) Compute the residual norm $\rho_j = \|F + J\delta^{(j)}\|_2$, of the solution $\delta^{(j)}$ that would be obtained if we stopped at this step.
  (c) If $\rho_j \le \epsilon_n$ set $m = j$ and go to (3).

(3) *Form the approximate solution:*
Arnoldi: Define H_m to be the $m \times m$ (Hessenberg) matrix whose nonzero entries are the coefficients h_{ij}, $1 \le i \le j$, $1 \le j \le m$ and define $V_m \equiv [v_1, v_2, \cdots, v_m]$.
- Find the vector y_m that solves the linear system $H_m y = \beta e_1$, where $e_1 = [1, 0, \cdots, 0]^T$.
- Compute $\delta^{(m)} = \delta^{(0)} + z^{(m)}$, where $z^{(m)} = V_m y_m$, and $u_{n+1} = u_n + \delta^{(m)}$.

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

(4) *Stopping test:* If u_{n+1} is determined to be a good enough approximation to a root of (2.1), then stop, else set $u_n \leftarrow u_{n+1}$, $n \leftarrow n+1$, choose a new tolerance ϵ_n, and go to (2).

Therefore, in both Arnoldi and GMRES the outer iteration is of the form $u_{n+1} = u_n + \delta^{(m)}$ where $\delta^{(m)} = \delta^{(0)} + z^{(m)}$, with

$$z^{(m)} = V_m y_m,$$

and y_m is either the solution of an $m \times m$ linear system, for Arnoldi, or the solution of an $(m+1) \times m$ least squares problem for GMRES.

Steps (2) and (3) of the above algorithm are precisely the Arnoldi (GMRES) method for solving the linear system $J\delta = -F$ (see [22] and [24]). Each outer loop of the above algorithm, consisting of steps (2), (3), and (4), is divided into two main stages. The first stage is an Arnoldi process, which builds an orthonormal basis $V_m = [v_1, v_2, \cdots, v_m]$ of the Krylov subspace K_m. If we denote by V_j the $N \times j$ matrix with column vectors $v_1, v_2, \cdots, v_j$, then it follows immediately from (2.7) that (see also [22] and [24])

$$JV_m = V_m H_m + \hat{v}_{m+1} e_m^T, \tag{2.8}$$

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

$$JV_m = V_{m+1} \bar{H}_m, \tag{2.9}$$

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

Step (3) of Newton–GMRES computes the approximate solution $\delta^{(m)}$ in $\delta^{(0)}+K_m$ that solves (2.6). This is accomplished by first letting $z = V_m y$ for $y \in \mathbf{R}^m$. Then $\|r^{(0)} - Jz\|_2 = \|\beta v_1 - JV_m y\|_2$. Using (2.9), we have

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

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

$$\min_{y\in\mathbf{R}^m} \|\beta e_1 - \bar{H}_m y\|_2. \tag{2.10}$$

Then the optimal δ is given by $\delta^{(m)} = \delta^{(0)} + V_m y_{GM}$. Note that (2.10) is a least squares problem of size $m + 1$, and its coefficient matrix is upper Hessenberg. The next iterate u_{n+1} is then computed at the end of step (3), by adding $\delta^{(m)}$. Finally, the stopping criteria in step (4) will be discussed in the numerical testing section.

For simplicity, we have omitted several details on the practical implementation of the above methods, which are discussed at length in [22], [4], and [24]. For example, the residual norm ρ_j referred to in step (2) of the algorithms does not require the computation of the approximate solution $\delta^{(j)}$ at every step. Instead an inexpensive formula, which evaluates ρ_j, is updated at each step while the factorization of the Hessenberg matrix H_m or $\bar{H}_m$ is updated (see [4] and [24] for details).

The Arnoldi algorithm is theoretically equivalent to the Conjugate Gradient method when J is symmetric and positive definite, and to the Lanczos method for solving linear systems when J is symmetric [22]. The GMRES algorithm is theoretically equivalent to GCR [8] and to ORTHODIR [15] but is less costly both in terms of storage and arithmetic [24]. For a synthesis and general description of available conjugate gradient type methods see [23]. A comparison of the cost of each step of these algorithms shows that for large enough m, GMRES costs about 1/3 less than GCR/ORTHOMIN in arithmetic, whereas storage is roughly divided by a factor of two. Another appealing property of GMRES is that in exact arithmetic, the method does not break down or, to be more accurate, it can only break down when it delivers the exact solution [24].

We note that if either algorithm solves the linear system $J\delta = -F(u)$ exactly, or rather with sufficient accuracy by taking (for example) m sufficiently large, then it is clear that the resulting algorithm is nothing but Newton's method, in which the Jacobian linear systems are solved by either Arnoldi or GMRES.

Perhaps one of the most important aspects of the above Krylov methods is that the Jacobian matrix J is never needed explicitly. The only operations with the Jacobian matrix J that are required from the Arnoldi process are matrix-vector multiplications $w = Jv$, which can be approximated by

$$J(u)v \approx \frac{F(u+\sigma v) - F(u)}{\sigma}, \tag{2.11}$$

where u is the point at which the Jacobian is being evaluated and σ is some carefully chosen small scalar. The idea of exploiting the above approximation is not new and was extensively used in the context of ODE methods [3]–[5], [11], [17], in eigenvalue calculations [9], [16] and is quite common in nonlinear equation solution methods and optimization methods (see, for example, [12],[19],[27]).

Another aspect of the above algorithms we have not yet considered is the ability to use restarting in the (linear) Krylov methods. In a typical implementation of the above Krylov methods, a maximum value of m is dictated by storage considerations. If we let $m_{\max}$ be this value, then it is possible that $m = m_{\max}$ in the Arnoldi process, and yet ρ_m is still greater than ϵ_n. In this case, we can set $\delta^{(0)}$ equal to $\delta^{(m)}$ and restart the Arnoldi process, effectively restarting the Krylov method. The convergence of such a procedure is not always guaranteed, but the idea seems to work well in practice. We note that for lack of a better initial guess we use $\delta^{(0)} = 0$ on the first (and possibly only) pass through the Arnoldi process at each stage of the Newton iteration. It is only when restarting that $\delta^{(0)}$ will be nonzero. As will be seen below, it will also be important to choose the tolerance ϵ_n at each step of the Newton iteration.

Finally, we note that as m becomes large, a considerable amount of the work involved is in making the vector v_{j+1} orthogonal to all the previous vectors $v_1, \cdots, v_j$. Saad [22] and Brown and Hindmarsh [4] have proposed incomplete versions of Arnoldi and GMRES, respectively, in which the vector v_{j+1} is only required to be orthogonal to the previous p vectors , $v_{j-p+1}, \cdots, v_j$. Equations (2.8) and (2.9) still hold in this case but the basis $V_{m+1} = [v_1, \cdots, v_{m+1}]$ is only partially orthogonal in the sense defined above. These algorithms are referred to as the *Incomplete Orthogonalization Method* (IOM) and IGMRES, respectively, and can be more cost effective than the complete methods on some problems. See [22] and [4] for details.

3. Linesearch backtracking techniques. Newton's method by itself may often fail to converge if the initial guess u_0 is far away from a root of (1.1). To enhance the robustness of the nonlinear Krylov algorithms considered in the previous section we will consider two modifications of these methods. In this section, we will consider a global strategy based on a linesearch backtracking procedure, and then in the next section we will investigate a model trust region approach.

Dennis and Schnabel [7] suggest using a global strategy for finding a root u_* of (1.1) that is based upon a globally convergent method for the problem

$$\min_{u \in \mathbf{R}^N} f(u) = \frac{1}{2} F(u)^T F(u). \tag{3.1}$$

A *descent direction* for f at the current approximation u is any vector p such that

$$\nabla f(u)^T p < 0,$$

where $\nabla f(u) = (\partial f / \partial u_1(u), \cdots, \partial f / \partial u_N(u))^T$. An easy calculation shows that

$$\nabla f(u) = J(u)^T F(u),$$

and so p is a descent direction for f at u if

$$F(u)^T J(u) p < 0.$$

For such a direction, one can show that there exists a certain $\lambda_0 > 0$ such that $f(u + \lambda p) < f(u)$ for all $0 < \lambda \leq \lambda_0$.

If $\bar{\delta}$ is an approximate solution of (2.2), with $F = F(u)$ and $J = J(u)$, then

$$F^T J \bar{\delta} = -F^T F - F^T \bar{r}, \tag{3.2}$$

where $\bar{r} = -F - J\bar{\delta}$ is the residual associated with $\bar{\delta}$. Thus, $\bar{\delta}$ will be a descent direction for f at u whenever $|F^T \bar{r}| < F^T F$. In particular, if $\|\bar{r}\|_2 < \|F\|_2$, then $\bar{\delta}$ is a descent direction. We have the following results regarding the existence of descent directions when using the Arnoldi and GMRES methods.

PROPOSITION 3.1. *Let u be the current Newton–Arnoldi iterate, $F \equiv F(u)$ and $J \equiv J(u)$. Assume J is nonsingular. Let $\delta^{(m)} = -V_m H_m^{-1} V_m^T F$ be the direction provided by the Arnoldi method assuming the initial guess $\delta^{(0)} = 0$. If $\delta^{(m)}$ exists, then it is a descent direction for f at u, and*

$$F^T J\delta^{(m)} = -F^T F, \tag{3.3}$$

for any $m = 1, \cdots, N$.

Proof. (See Brown [2, Thm. 3.5]). This result follows from the equality (3.2) and the fact that the residual vector in Arnoldi's method is orthogonal to the Krylov subspace, and in particular to its first basis vector, which is F up to a constant factor. □

PROPOSITION 3.2. *Let u be the current Newton–*GMRES *iterate, $F = F(u)$ and $J = J(u)$. Assume J is nonsingular. Let $\delta^{(m)} = V_m y_{GM}$ be the direction provided by the* GMRES *method assuming the initial guess $\delta^{(0)} = 0$. If $\delta^{(m)} \neq 0$, then $\delta^{(m)}$ is a descent direction for f at u, and*

$$F^T J\delta^{(m)} = -F^T F + \rho_m^2, \tag{3.4}$$

for any $m = 1, \cdots, N$, where ρ_m is the residual norm achieved at the mth step of the GMRES *algorithm.*

Proof. We first prove (3.4). According to (3.2) we only have to show that $\rho_m^2 = -F^T r^{(m)}$ with $r^{(m)} = -F - J\delta^{(m)}$. By definition (see (2.6)), *GMRES* minimizes $\|F + Jz\|_2$ for z in the Krylov subspace K_m. It is known in this case that the residual vector corresponding to the minimizer is orthogonal to JK_m (see, e.g., [23]). Therefore, we have

$$(r^{(m)}, F) = (r^{(m)}, -r^{(m)} - J\delta^{(m)}) = -\rho_m^2,$$

which establishes (3.4). The fact that $\delta^{(m)}$ is a descent direction in this case, has been shown in Theorem 3.7 of [2]. It can also be shown from (3.4) and the definition (2.6), from which we get that $\rho_m < \rho_0 = \|F\|_2$, since $\delta^{(m)} \neq 0$. □

We also have the following result regarding the existence of descent directions in the subspace K_m for f at u.

PROPOSITION 3.3. *Let u be the current Newton–*GMRES *iterate, $F = F(u)$ and $J = J(u)$. Assume J is nonsingular. Let $\delta^{(m)} = V_m y_{GM}$ be the direction provided by the* GMRES *method assuming the initial guess $\delta^{(0)} = 0$. Then there exist descent directions in the subspace K_m for the function f at u if and only if*

$$\rho_m = \|F + J\delta^{(m)}\|_2 < \|F\|_2 \tag{3.5}$$

holds.

Proof. This result is clear from Proposition 3.2 and from the fact that the GMRES iterate solves the minimization problem (2.6). □

The assumption that J is nonsingular in the above propositions is necessary, since Arnoldi and GMRES are only guaranteed to converge for nonsingular systems. When J is singular, the Arnoldi process can break down before providing any useful information, since Jv_i might vanish for some i. Furthermore, while Arnoldi's method will converge in at most N iterations for any nonsingular J, the Hessenberg matrix

H_m may be singular for some $m < N$, and the mth Arnoldi iterate may not exist as a result, i.e., there may be no solution to the system $H_m y = \beta e_1$. See [4], [22], and [24] for more details.

Condition (3.5) means that the residual norm in GMRES must be reduced strictly. It holds whenever the Jacobian matrix is positive real and at least one step of GMRES is performed, i.e., $m \geq 1$ (see Brown [2] and Elman [8]). The condition that J be positive real at every step is too strong a condition to require. A milder condition is to assume that the dimension m in Arnoldi or GMRES is large enough to ensure that the final residual is reduced by a factor of at least η, where η is a scalar < 1. In other words,

$$\|J\delta + F\|_2 \leq \eta\|F\|_2, \tag{3.6}$$

where $\eta < 1$, and δ is the Arnoldi or GMRES iterate.

The practicality of the assumption that m can be chosen as large as necessary to guarantee that (3.6) holds is questionable. It is known that as m gets large, (3.6) will eventually be fulfilled. The problem is that with m too large, computational cost and storage become too high. An alternative would be to perform restarting within the (linear) Arnoldi or GMRES algorithm itself. In this case, however, the above results (3.3) and (3.4) do not hold, since their proofs rely on the assumption that $\delta^{(0)} = 0$, while restarting has the same effect as making the initial guess nonzero in Arnoldi or GMRES. To derive similar results, we go back to (3.2), which becomes

$$\nabla f(u)^T \delta^{(m)} = -F^T F - F^T r^{(m)}. \tag{3.7}$$

Thus, for both GMRES and Arnoldi, it suffices that $\delta^{(m)}$ will be a descent direction whenever $\|r^{(m)}\|_2 < \|F\|_2$, a condition that has already been seen at the beginning of this section.

We also need to obtain alternative expressions for $\nabla f(u)^T \delta^{(m)}$ for computational purposes. It is easy to show (see, e.g., [22]) that for Arnoldi's method the residual vector satisfies

$$r^{(m)} = -F - J\delta^{(m)} = -(e_m^T y_m)\hat{v}_{m+1},$$

which yields from (3.7),

$$\nabla f(u)^T \delta^{(m)} = -F^T F + (e_m^T y_m) F^T \hat{v}_{m+1}. \tag{3.8}$$

Note that when $\delta^{(0)} = 0$ the last term in the above expression vanishes, since then $F = -\beta v_1$, and the columns of V_m are orthogonal to $\hat{v}_{m+1}$. For GMRES we may use (3.7) but we need to compute $r^{(m)}$. We have,

$$\begin{aligned} r^{(m)} &= -F - J\delta^{(m)} \\ &= r^{(0)} - JV_m y_m \\ &= V_{m+1}(\beta e_1 - \bar{H}_m y_{GM}), \end{aligned} \tag{3.9}$$

which does not involve function evaluations. It can also be shown that (see [4] for details)

$$\beta e_1 - \bar{H}_m y_{GM} = \beta q_{m+1} q_{m+1}^T e_1,$$

where q_{m+1} is the last column of the Q matrix in the QR factorization of $\bar{H}_m$.

Our backtracking procedure will be based upon the ideas presented by Dennis and Schnabel [7]. Given the current Newton iterate $u = u_n$ and a descent direction p, we want to take a step in the direction of p that yields an acceptable u_{n+1}. We will define a step $\delta = \lambda p$ to be *acceptable* if the following Goldstein–Armijo [7] conditions are met:

$$f(u + \lambda p) \leq f(u) + \alpha\lambda\nabla f(u)^T p, \text{ and} \tag{3.10}$$

$$f(u + \lambda p) \geq f(u) + \beta\lambda\nabla f(u)^T p, \tag{3.11}$$

for given scalars α, β satisfying $0 < \alpha < \beta < 1$. These two conditions are commonly referred to as the α- and β-conditions, respectively. For a given descent direction p, the next result shows that there exist points $u + \lambda p$ satisfying (3.10) and (3.11).

THEOREM 3.4. *Let $F : \mathbf{R}^N \to \mathbf{R}^N$ be continuously differentiable on $\mathbf{R}^N$. Let $f(u) = \frac{1}{2}F(u)^T F(u)$, and u, p in $\mathbf{R}^N$ such that $\nabla f(u)^T p < 0$. Then given $0 < \alpha < \beta < 1$, there exist $\lambda_u > \lambda_\ell > 0$ such that $u + \lambda p$ satisfies* (3.10) *and* (3.11) *for any $\lambda \in (\lambda_\ell, \lambda_u)$.*

Proof. This is essentially Theorem 6.3.2 of Dennis and Schnabel [7, p. 120]. □

We next present the particular backtracking algorithm we have chosen to use. The selection procedure for λ is modeled after that in [7].

Algorithm: Linesearch Backtrack

```
(1) Choose $\alpha \in (0, \frac{1}{2})$ and $\beta \in (\frac{1}{2}, 1)$.
(2) Given $u_n$ the current Newton iterate, calculate $p = \delta^{(m)}$, where $\delta^{(m)} = \delta^{(0)} + z^{(m)}$, and $z^{(m)} = V_m y_m$. Here, $y_m$ is calculated using either the Arnoldi or GMRES method (with or without restarting), and it is assumed that (3.6) holds with $\delta = \delta^{(m)}$.
(3) Calculate $\nabla f(u_n)^T p$ using the appropriate choice(s) from equations (3.3), (3.4), (3.7), (3.8), and (3.9).
(4) Find an acceptable new iterate $u_{n+1} = u_n + \lambda p$. First, set $\lambda = 1$. Define $u(\lambda) = u_n + \lambda p$.
    (a) If $u(\lambda)$ satisfies (3.10) and (3.11), then exit. If not, then continue.
    (b) If $u(\lambda)$ satisfies (3.10), but not (3.11), and $\lambda \geq 1$, set $\lambda \leftarrow 2\lambda$ and go to (a).
    (c) If $u(\lambda)$ satisfies (3.10) only and $\lambda < 1$, or $u(\lambda)$ does not satisfy (3.10) and $\lambda > 1$, then
        (c.1) If $\lambda < 1$, define $\lambda_{lo} = \lambda$ and $\lambda_{hi}$ = last previously attempted value of $\lambda$. If $\lambda > 1$, define $\lambda_{lo}$ = last previously attempted value of $\lambda$ and $\lambda_{hi} = \lambda$. In both cases, $u(\lambda_{lo})$ satisfies (3.10) but not (3.11), $u(\lambda_{hi})$ does not satisfy (3.10), and $\lambda_{lo} < \lambda_{hi}$.
        (c.2) Find $\lambda \in (\lambda_{lo}, \lambda_{hi})$ such that $u(\lambda)$ satisfies (3.10) and (3.11) using successive linear interpolation.
    (d) Otherwise ($u(\lambda)$ does not satisfy (3.10) and $\lambda \leq 1$), decrease $\lambda$ by a factor between 0.1 and 0.5 as follows:
        (d.1) Select the new $\lambda$ such that $u(\lambda)$ is the minimizer of the one-dimensional quadratic interpolant passing through $f(u_n)$, $f'(u_n) = \nabla f(u_n)^T p$ and $f(u_n + \lambda p)$. Then take the maximum of this new $\lambda$ and 0.1 as the actual value used. (One can show theoretically that the new $\lambda$ value so chosen will be less than or equal to one-half the previous value.)
        (d.2) Go to step (b).
```

We note that the current algorithm will likely break down if the Jacobian J is singular. For now, all that can be done in this case is to restart the iteration with a different initial guess u_0.

It remains to discuss the choice of ϵ_n. From the results in [6], choosing $\epsilon_n = \eta\|F(u_n)\|_2$, where $0 < \eta < 1$, guarantees the linear convergence of the Newton–Krylov iteration for a good enough initial guess. If η is replaced by a sequence η_n decreasing to zero and satisfying $0 \leq \eta_n < 1$ for all n, then the iteration converges superlinearly. It is possible to show that for this second choice of ϵ_n the full Arnoldi or GMRES step will be admissible near a root (i.e., that conditions (3.10) and (3.11) will be satisfied with $\lambda = 1$). However, a finer analysis may indicate that the looser tolerance may in fact give convergence of the iteration.

4. Model trust region techniques.

In this section we will propose a model trust region strategy in connection with the Newton–GMRES algorithm. In particular, we will consider a dogleg strategy based on Powell's hybrid method [21].

Let u be the current approximate solution of (1.1). The effect of using a Krylov method to solve the Newton equations (1.2) approximately is to take a step from u of the form $u + \delta$, where δ is in the affine subspace $\delta^{(0)} + K_m$. If $V_m = [v_1, \cdots, v_m]$ is an orthonormal basis for K_m, and the initial guess $\delta^{(0)} = 0$, then $\delta = V_m y$, for some $y \in \mathbf{R}^m$, and we have a step of the form $u + V_m y$. Hence, we are effectively restricting our search directions from u to be in the subspace K_m.

4.1. Global strategy with restriction to a subspace.

Our global strategy will again be based upon finding a local minimum of the real-valued function $f(u) = \frac{1}{2}F(u)^T F(u)$. Thus, we want to solve the minimization problem

$$\min_{y \in \mathbf{R}^m} f(u + V_m y).$$

Letting $g(y) = f(u + V_m y)$, we then have

$$\nabla g(y) = (J(u + V_m y)V_m)^T F(u + V_m y),$$

and in particular that

$$\nabla g(0) = (JV_m)^T F,$$

where $F = F(u)$ and $J = J(u)$.

If we use $F + JV_m y$ as a linear model of $F(u + V_m y)$, then a natural quadratic model for g is

$$\hat{g}(y) = \frac{1}{2}\|F + JV_m y\|_2^2. \tag{4.1}$$

Letting $B_m = V_m^T J^T J V_m$, we have

$$\hat{g}(y) = \frac{1}{2}F^T F + F^T J V_m y + \frac{1}{2} y^T B_m y,$$

where B_m is symmetric and positive semidefinite, and $\nabla\hat{g}(0) = \nabla g(0)$. If J is nonsingular, then B_m is positive definite, since V_m has orthonormal columns. Our *model trust region* approach will be based upon trying to find a solution of the problem

$$\min_{\|y\|_2 \leq \tau} \hat{g}(y), \quad y \in \mathbf{R}^m, \tag{4.2}$$

where τ is an estimate of the maximum length of a successful step we are likely to be able to take from u. It is also a measure of the size of the region in which the local quadratic model $\hat{g}(y)$ closely agrees with the function $g(y)$. The solution to (4.2) is given in the next lemma.

LEMMA 4.1. *Let $\hat{g}(y)$ be defined by* (4.1), *and assume that J is nonsingular. Then problem* (4.2) *is solved by*

$$y_m(\mu) = (B_m + \mu I)^{-1} d_m, \tag{4.3}$$

where $d_m = -\nabla \hat{g}(0)$, for the unique μ such that $\|y_m(\mu)\|_2 = \tau$, unless $\|y_m(0)\|_2 \leq \tau$, in which case $y_m(0) = B_m^{-1} d_m$ is the solution. Furthermore, for any $\mu \geq 0$, $\delta(\mu) = V_m y_m(\mu)$ defines a descent direction for $f(u) = \frac{1}{2} F(u)^T F(u)$ from u, as long as $d_m \neq 0$.

Proof. Since J being nonsingular implies that B_m is positive definite, (4.3) follows from Lemma 6.4.1 of Dennis and Schnabel [7, p. 131] and $y_m(\mu)$ is the unique solution to (4.2). It therefore only remains to show that $\delta(\mu) = V_m y_m(\mu)$ is a descent direction for $f(u)$ at u, for all $\mu \geq 0$. Recall that p is a descent direction for f at u if

$$\nabla f(u)^T p < 0,$$

or since $\nabla f(u) = J^T F$, if

$$F^T J p < 0.$$

For $p = \delta(\mu)$, we have

$$\begin{aligned} F^T J \delta(\mu) &= F^T J V_m (B_m + \mu I)^{-1} d_m \\ &= ((J V_m)^T F)^T (B_m + \mu I)^{-1} d_m \\ &= -d_m^T (B_m + \mu I)^{-1} d_m \\ &< 0, \end{aligned}$$

since $d_m = -\nabla \hat{g}(0) = \nabla g(0) \neq 0$, and since $B_m + \mu I$ is positive definite for all $\mu \geq 0$. □

Since there is no finite method of determining μ such that $\|y_m(\mu)\|_2 = \tau$ when $\tau < \|B_m^{-1} d_m\|_2$, we only approximately solve (4.2). The *dogleg strategy* of Powell [21] makes a piecewise linear approximation to the curve $y_m(\mu)$, and takes $\hat{y}_m$ as the point on this curve for which $\|\hat{y}_m\|_2 = \tau$. We then define $u_{n+1} = u_n + \hat{\delta}$, where $\hat{\delta} = V_m \hat{y}_m$. If the iterate u_{n+1} is acceptable, in the sense that some α-condition (3.10) is satisfied, we proceed to the next step, while if not, a new value of the trust region size τ is chosen, and the procedure is repeated.

4.2. Global strategies for GMRES. The preceding discussion is independent of the choice of the subspace K_m, i.e., the results are valid for any subspace K_m of dimension m with orthonormal basis given by the columns of V_m. In fact, the basis vectors v_i $(i = 1, \cdots, m)$ need not even form an orthogonal basis for K_m. We next turn our attention to the case in which K_m and V_m are generated using the GMRES algorithm. Here, the relation (2.9) allows an easy reformulation of the minimization problem (4.2). First, note that when the initial guess is $\delta^{(0)} = 0$ and m steps of the Arnoldi process have been taken in the Newton–GMRES algorithm we have

$$d_m = -\nabla \hat{g}(0) = -(J V_m)^T F = -\beta \bar{H}_m^T e_1. \tag{4.4}$$

This direction is referred to as the *steepest descent direction* for $\hat{g}(y)$ at $y = 0$, and is the same as that for $g(y)$. Next,

$$\begin{aligned} B_m &= (JV_m)^T JV_m \\ &= (V_{m+1}\bar{H}_m)^T V_{m+1}\bar{H}_m \\ &= \bar{H}_m^T \bar{H}_m. \end{aligned} \tag{4.5}$$

Here we have used the relation (2.9), which is satisfied by the Arnoldi vectors v_i. Thus, $\hat{g}(y)$ can be rewritten as

$$\hat{g}(y) = \frac{1}{2}F^T F + \beta e_1^T \bar{H}_m y + \frac{1}{2} y^T \bar{H}_m^T \bar{H}_m y. \tag{4.6}$$

Furthermore, the fact that J is of full rank implies that $\bar{H}_m$ is also of full rank by (2.9) and the orthogonality of V_m and V_{m+1}. Note that this is true even when $h_{m+1,m} = 0$, because in this case the relation (2.9) reduces to $JV_m = V_m H_m$, with H_m nonsingular and $\bar{H}_m^T \bar{H}_m = H_m^T H_m$.

Minimizing $\hat{g}(y)$ in the steepest descent direction amounts to minimizing

$$\hat{g}(\alpha d_m) = \frac{1}{2}F^T F + \alpha\beta e_1^T \bar{H}_m d_m + \frac{\alpha^2}{2}\|\bar{H}_m d_m\|_2^2.$$

The optimal value for α is

$$\alpha_{\text{opt}} = \frac{\|d_m\|_2^2}{\|\bar{H}_m d_m\|_2^2},$$

which is defined as long as $d_m \neq 0$, since $\bar{H}_m$ has full column rank. We refer to the point where $\hat{g}(\alpha d_m)$ assumes its minimum value as the *Cauchy Point*, and write

$$y_{CP} = \alpha_{\text{opt}} d_m = -\beta \frac{\|d_m\|_2^2}{\|\bar{H}_m d_m\|_2^2} \bar{H}_m^T e_1. \tag{4.7}$$

We note that GMRES gives the global minimizer of $\hat{g}(y)$, and we write

$$y_{GM} = -(\bar{H}_m^T \bar{H}_m)^{-1} \beta \bar{H}_m^T e_1 = (\bar{H}_m^T \bar{H}_m)^{-1} d_m. \tag{4.8}$$

Note that in practice the above formula will never be used explicitly. Numerically, it is more satisfactory to compute y_{GM} as the solution of the the least squares problem $\min \|\bar{H}_m y + \beta e_1\|_2$ over the variable y. We should point out here that when $h_{m+1,m} = 0$ the determination of y_{GM} in this manner does not cause any difficulty. In fact, in this situation we will have what is called a *happy breakdown* in GMRES, in that $\delta = V_m y_{GM}$ becomes the exact solution to the linear system $J\delta = -F$, i.e., we obtain a full Newton step instead of an inexact Newton step. The GMRES solution and the Arnoldi solution are then identical.

In order to use the dogleg strategy, we have seen that the vector d_m must be nonzero. Since d_m is the steepest descent direction within the Krylov subspace, a null value would indicate that there are no descent directions from f at u within the subspace K_m. Note that (4.8) implies $y_{GM} = 0 \Leftrightarrow d_m = 0$. Hence, if we require m to be large enough so that condition (3.6) holds, then necessarily y_{GM} will be nonzero, and the dogleg strategy can be used. This will be assumed in the dogleg algorithm presented below.

We next describe the dogleg algorithm for $\hat{g}(y)$. Suppose we are in a situation where the full GMRES step is unsatisfactory. This indicates that the quadratic model $\hat{g}(y)$ does not adequately model $g(y)$ in a region containing the full GMRES step. The linesearch algorithm of the previous section would take a step in the same direction, but of shorter length. In the dogleg strategy, we first choose a shorter steplength, and then use the full m-dimensional quadratic model $\hat{g}(y)$ to choose the new direction. The curve $y_m(\mu)$ given by (4.3) is approximated by the piecewise linear curve from zero to y_{CP}, and from y_{CP} to y_{GM}. As indicated above, given a trust region size τ, we then find the point $\hat{y}_m$ on the dogleg curve with $\|\hat{y}_m\|_2 = \tau$.

The only remaining parts of the algorithm to discuss are the decision process for an acceptable new iterate u_{n+1} and the selection of the trust region size τ. We will again base our strategy on the ideas presented in [7]. The condition for accepting u_{n+1} is (3.10), namely,

$$f(u_n + \hat{\delta}) \leq f(u_n) + \alpha \nabla f(u_n)^T \hat{\delta},$$

where $\hat{\delta} = V_m \hat{y}_m$. In this case, the relations (3.3) and (3.4) do not apply, since $\hat{\delta}$ is not necessarily the Arnoldi or GMRES direction. However, for any $\delta = V_m y$, we have

$$\nabla f(u_n)^T \delta = F^T J \delta = F^T J V_m y = F^T V_{m+1} \bar{H}_m y = -\beta e_1^T \bar{H}_m y. \tag{4.9}$$

Hence, it will be easy to determine if (3.10) is satisfied for any u_{n+1} of the form $u_n + \hat{\delta}$ without even computing the direction $\hat{\delta}$.

The trust region size will be adjusted based upon a comparison of the two values

$$\Delta f \equiv f(u_{n+1}) - f(u_n) = g(\hat{y}_m) - g(0),$$

which is the *actual reduction* in the function f, and

$$\Delta f_{\text{pred}} \equiv \hat{g}(\hat{y}_m) - g(0),$$

which is the *predicted reduction* in the function f. Our dogleg algorithm is then as follows.

Algorithm: Dogleg

(1) Choose $\alpha \in (0, \frac{1}{2})$.

(2) Given u_n, the current Newton iterate, calculate $\delta^{GM} = V_m y_{GM}$. Here, y_{GM} is calculated using the GMRES method (without restarting) with initial guess $\delta^{(0)} = 0$, and it is assumed m is large enough so that (3.6) holds with $\delta = \delta^{GM}$.

(3) Given τ, the current trust region size, calculate $\hat{y}_m$, the point on the dogleg curve for which $\|\hat{y}_m\|_2 = \tau$. Then calculate $u_{n+1} = u_n + V_m \hat{y}_m$. If u_{n+1} is acceptable, then go to step (5).

(4) If u_{n+1} is not acceptable, then do one of the following:
   (a) If τ has been doubled during this iteration, then set u_{n+1} equal to its last accepted value and set $\tau \leftarrow \tau/2$. Then continue to the next Newton iteration. If not:
   (b) Determine a new τ by using the minimizer of the one-dimensional quadratic interpolating $f(u_n)$, $f(u_{n+1})$, and the directional derivative of f at u_n in the direction $\hat{\delta} = V_m \hat{y}_m$. Letting λ be the value for which $u_n + \lambda \hat{\delta}$ is this minimizer, set $\tau \leftarrow \lambda \|\hat{\delta}\|_2$, but constraining it to be between 0.1 and 0.5 of the old τ. Then go to step (3).

(5) For an acceptable u_{n+1}, calculate Δf and Δf_{pred}. Then do one of the following:
   (a) If Δf and Δf_{pred} agree to within relative error 0.1, and τ has not been decreased during this iteration, set $\tau \leftarrow 2 * \tau$, and go to step (3). If not:
   (b) If $\Delta f > 0.1 * \Delta f_{\text{pred}}$ set $\tau = \tau/2$, or if $\Delta f < 0.75 * \Delta f_{\text{pred}}$, set $\tau = 2 * \tau$. Otherwise, do not change τ. Then continue to the next Newton iteration. (Note that here both Δf and Δf_{pred} are negative.)

We note that this algorithm will also likely break down if the Jacobian J is singular. As with the linesearch technique, all that can be done for now is to restart the iteration with a different initial guess u_0. We also set $\epsilon_n = \eta_n \|F(u_n)\|_2$ as before.

Although we have based our trust region algorithm on the dogleg strategy outlined in [7], we could alternatively have based our approach on one of the related algorithms for unconstrained optimization presented by Moré [18] and Gay [10]. In general, for unconstrained optimization problems the matrix B_m can have negative eigenvalues. In this case, Gay [10] has shown that the solution of (4.2) is still a y_* value satisfying $(B_m + \mu I)y_* = d_m$ for some $\mu > 0$ such that $B_m + \mu I$ is positive semidefinite.

4.3. Restarting procedures. One can allow restarting in the (linear) Krylov method in the above algorithm. Equivalently, we would like to show how to use a nonzero initial vector $\delta^{(0)}$ in the algorithm. In this case, the step δ from u would normally have the form $\delta = \delta^{(0)} + V_m y$. However, with this choice $g(y) = f(u+\delta) = f(u + \delta^{(0)} + V_m y)$ would not necessarily be close to $f(u)$ for small y, as it should be. That is, $g(0) \neq f(u)$ in general. For this reason, the contribution to the step from the initial guess must also be variable. This has the effect of enlarging the dimension of the minimization problem (4.2). For $\delta^{(0)}$ a nonzero initial guess, let $\bar{y} = (y, t)^T \in \mathbf{R}^{m+1}$ and try taking a step from u of the form $u+\delta$, where $\delta = [V_m, \delta^{(0)}]\bar{y}$. (Here, we assume $V_m = [v_1, \cdots, v_m]$ has been generated by taking m steps of the Arnoldi process.) Then we have

$$F(u+\delta) \approx F + J[V_m, \delta^{(0)}]\bar{y}.$$

Letting $W = [V_m, \delta^{(0)}]$, the local quadratic model for $g(\bar{y}) = f(u + W\bar{y}) = \frac{1}{2}\|F(u + W\bar{y})\|_2^2$ is now given by

$$\hat{g}(\bar{y}) = \frac{1}{2}\|F + JW\bar{y}\|_2^2.$$

Since $r^{(0)} = -F - J\delta^{(0)}$, we can write

$$F + JW\bar{y} = (1-t)F - t\, r^{(0)} + V_{m+1}\bar{H}_m y,$$

using the fact that $JV_m = V_{m+1}\bar{H}_m$. This then makes it possible to evaluate $\hat{g}(\bar{y})$ without the need for the Jacobian matrix J.

For this modified quadratic model, the *steepest descent direction* is given by

$$\begin{aligned} d &= -\nabla\hat{g}(0) = -(JW)^T F \\ &= -[JV_m, J\delta^{(0)}]^T F \\ &= -[V_{m+1}\bar{H}_m, J\delta^{(0)}]^T F \\ &= \begin{pmatrix} -\bar{H}_m^T V_{m+1}^T F \\ (F + r^{(0)})^T F \end{pmatrix} \in \mathbf{R}^{m+1}. \end{aligned}$$

Next, the *approximate Newton step* is found using a relationship between the Newton step and the steepest descent direction similar to the situation earlier without restarting, namely,

$$s = B^{-1}d,$$

where $B = \nabla^2 \hat{g}(0)$ is the Hessian of $\hat{g}$. An easy calculation gives

$$B = (JW)^T(JW) = \begin{pmatrix} \bar{H}_m^T \bar{H}_m & v \\ v^T & a \end{pmatrix},$$

letting $v = -\bar{H}_m^T V_{m+1}^T(F + r^{(0)})$ and $a = \|F + r^{(0)}\|_2^2$. To calculate s, first note that we already have $\bar{H}_m = QR$, the QR-factorization of $\bar{H}_m$. Hence,

$$B = \begin{pmatrix} R^T R & v \\ v^T & a \end{pmatrix}.$$

Letting

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

where $\bar{R} \in \mathbf{R}^{m \times m}$, we have $R^T R = \bar{R}^T \bar{R}$, which is essentially a Cholesky decomposition of $\bar{H}_m^T \bar{H}_m$. Note that the matrix $\bar{R}$ is always nonsingular for J nonsingular, regardless of the value of $h_{m+1,m}$ in the Arnoldi process. The Cholesky decomposition of B is then easily obtained as

$$B = \begin{pmatrix} \bar{R}^T & 0 \\ w^T & b \end{pmatrix} \begin{pmatrix} \bar{R} & w \\ 0 & b \end{pmatrix} \equiv C^T C, \tag{4.10}$$

where $w = (\bar{R}^T)^{-1} v$ and $b = \sqrt{a - w^T w}$.

The factorization (4.10) is possible as long as B is positive semidefinite. This follows immediately from its definition, since when J is nonsingular the matrix $B = (JW)^T JW$ is nonsingular if and only if the columns of JW are linearly independent, or equivalently if and only if the columns of W are linearly independent. This will be the case if and only if $\delta^{(0)}$ does not belong to K_m. In the situation where $\delta^{(0)}$ does belong to K_m, $\hat{g}(\bar{y})$ has an infinite number of global minimizers, and B is singular.

To handle the problem of an ill-conditioned or singular B, we use a technique similar to that done in the full N-dimensional quadratic approximation. If C is singular, or its condition number is greater than $1/\sqrt{\gamma}$, where γ is the machine epsilon, then we perturb the quadratic model $\hat{g}(\bar{y})$ to

$$\begin{aligned} h(\bar{y}) &= \hat{g}(\bar{y}) + \frac{1}{2}\mu \bar{y}^T \bar{y} \\ &= \frac{1}{2}F^T F + F^T(JW\bar{y}) + \frac{1}{2}\bar{y}^T(B + \mu I)\bar{y} \\ &= \frac{1}{2}F^T F - d^T \bar{y} + \frac{1}{2}\bar{y}^T(B + \mu I)\bar{y}, \end{aligned}$$

where $\mu = \sqrt{N \cdot \gamma}\,\|B\|_1$. The condition number of $B + \mu I$ is roughly $1/\sqrt{\gamma}$ (see p. 151 in [7] for a justification of this fact). Lemma 4.1 implies that the Newton step to the global minimizer of $h(\bar{y})$, $s = -(B + \mu I)^{-1} d$, solves the minimization problem

$$\min_{\|\bar{y}\|_2 \le \tau} \hat{h}(\bar{y}), \quad \bar{y} \in \mathbf{R}^{m+1},$$

for some $\tau > 0$. It follows easily that this step will also be a descent direction for $f(u)$.

The above procedure is similar to choosing the approximate Newton step s by making it the minimum 2-norm solution of

$$\min_{\bar{y}\in\mathbf{R}^{m+1}} \|F + JW\bar{y}\|_2. \tag{4.11}$$

To see this, let $(JW)^+$ be the pseudo-inverse of JW. Then the solution of (4.11) is $s_{LS} = -(JW)^+F$. For μ small, the step $\hat{s} = -(B+\mu I)^{-1}(JW)^TF$ is similar to s_{LS} in that $\hat{s} \to s_{LS}$ as $\mu \to 0$, and both are orthogonal to the null space of JW for any $\mu > 0$. Again, see [7] for more details.

5. Scaling and preconditioning.

The linear Krylov methods discussed in §2 need to be enhanced in order to improve their efficiency and robustness. Two ways of accomplishing this is by using scaling and preconditioning. Scaling is also of particular importance in solving systems of nonlinear equations, as Dennis and Schnabel note [7]. In this section we first discuss scaling of the nonlinear system, its effect on the linear systems to be solved, and then the use of preconditioning in solving the linear systems.

We will allow scaling of both the nonlinear function F and the unknown u in our algorithms. This will be accomplished by using two diagonal scaling matrices D_F and D_u with positive diagonal entries, where the scaled version of (1.1) is

$$\tilde{F}(\tilde{u}) = D_F F(D_u^{-1}\tilde{u}) = 0. \tag{5.1}$$

Here, $\tilde{F} = D_F F$ and $\tilde{u} = D_u u$ are the scaled versions of F and u. The resulting scaled Jacobian $\tilde{J}$ of $\tilde{F}$ is

$$\tilde{J}(\tilde{u}) = \tilde{F}'(\tilde{u}) = D_F J(D_u^{-1}\tilde{u})D_u^{-1}.$$

The scaled Newton equations are then

$$\tilde{J}\tilde{\delta} = -\tilde{F},$$

with $\tilde{\delta} = D_u\delta$. Since we will primarily be using finite-difference versions of the Krylov algorithms, the matrix J will not be available, and hence an explicit scaling of it cannot be performed. However, we will effectively perform an explicit scaling of the Newton equations without ever scaling J. For details of implementing scaling in this way using Krylov methods, see Brown and Hindmarsh [3].

The global strategies discussed in the previous two sections will work with the scaled problem (5.1). However, the scaling will be implemented in an implicit way as follows. Only the diagonal entries of D_F and D_u will be stored, and whenever one of the norms $\|\tilde{F}\|_2$ or $\|\tilde{u}\|_2$ is needed, a call will be made to a routine which computes the scaled norm. The scaled function $\tilde{f} = \frac{1}{2}\tilde{F}^T\tilde{F}$ will be the one used in determining the step from the current iterate to the next.

Without some form of preconditioning, the Krylov methods discussed in §2 are of limited use. Generally, by preconditioning here we mean applying the Krylov method to the equivalent system

$$(P_1^{-1}JP_2^{-1})(P_2\delta) = P_1^{-1}b \text{ or } \bar{J}\bar{\delta} = \bar{b}, \tag{5.2}$$

where $b = -F$. The matrices P_1 and P_2 are chosen in advance so that the preconditioned problem will be easier to solve than the original one. Since linear systems of the form $P_1 x = c$ and $P_2 x = c$ need to be solved, it is necessary that these additional linear systems be much easier to solve than the original problem. Also, preconditioning makes sense only if the convergence rate of the Krylov method applied to (5.2) is much better than that for the unpreconditioned problem.

The linear system (5.2) is said to be preconditioned on both sides, with P_1 and P_2 referred to as the left and right preconditioners, respectively. If one of these is the identity, then the system is said to be preconditioned on the left or right only.

Incorporating both scaling and preconditioning into the linear system, we then have the scaled preconditioned system

$$(D_F \tilde{J} D_u^{-1})(D_u \tilde{\delta}) = (D_F \tilde{b}) \text{ or } \tilde{\tilde{J}}\tilde{\tilde{\delta}} = \tilde{\tilde{b}}. \tag{5.3}$$

In our case, however, we will need to restrict P_1 to be the identity matrix. The norm of the residual associated with (5.3) is

$$\|\tilde{r}\|_2 = \|\tilde{\tilde{b}} - \tilde{\tilde{J}}\tilde{\tilde{\delta}}\|_2,$$

and for general P_1 this is not directly related to the quantity $\|\tilde{r}\|_2 = \|\tilde{b} - \tilde{J}\tilde{\delta}\|_2$ in which we are really interested. Furthermore, our global strategy is based upon the function $\tilde{f} = \frac{1}{2}\tilde{F}^T\tilde{F}$. Hence, the equations (3.3), (3.4) and the similar result used in the dogleg strategy will only hold when P_1 is the identity matrix. Letting $P_1 = I$ and $P_2 = P$, our scaled preconditioned linear system is then

$$(D_F J P^{-1} D_u^{-1})(D_u P \delta) = D_F b. \tag{5.4}$$

The scaled preconditioned Arnoldi and GMRES algorithms are then as follows.

Algorithm: SP–Arnoldi (SP–GMRES)

```
(1) *Arnoldi process:*
    • Choose a tolerance $\epsilon$.
    • For an initial guess $\delta^{(0)}$, form $r^{(0)} = -F - J\delta^{(0)}$, where $F = F(u)$ and
      $J = J(u)$.
    • Form $\tilde{r}^{(0)} = D_F r^{(0)}$. Compute $\tilde{\beta} = \|\tilde{r}^{(0)}\|_2$ and $\tilde{v}_1 = \tilde{r}^{(0)}/\tilde{\beta}$.
    • For $j = 1, 2, \cdots$, do:
      (a) Compute $\tilde{J}\tilde{v}_j = D_F(J(P^{-1}(D_u^{-1}\tilde{v}_j)))$.
      (b) $\tilde{h}_{i,j} = (\tilde{J}\tilde{v}_j, \tilde{v}_i), \quad i = 1, 2, \cdots, j$.
      (c) $\hat{v}_{j+1} = \tilde{J}\tilde{v}_j - \sum_{i=1}^{j} \tilde{h}_{i,j}\tilde{v}_i$.
      (d) $\tilde{h}_{j+1,j} = \|\hat{v}_{j+1}\|_2$, and
      (e) $\tilde{v}_{j+1} = \hat{v}_{j+1}/\tilde{h}_{j+1,j}$.
      (f) Compute the residual norm $\tilde{\rho}_j = \|\tilde{F} + \tilde{J}\tilde{\delta}^{(j)}\|_2$, of the solution $\tilde{\delta}^{(j)}$
          that would be obtained if we stopped at this step.
      (g) If $\tilde{\rho}_j \le \epsilon$ set $m = j$ and go to (2).
(2) *Form the approximate solution:*
    **Arnoldi:** Define $\tilde{H}_m$ to be the $m \times m$ (Hessenberg) matrix whose nonzero
    entries are the coefficients $\tilde{h}_{ij}$, $1 \le i \le j$, $1 \le j \le m$ and $\tilde{V}_m \equiv [\tilde{v}_1, \cdots, \tilde{v}_m]$.
    • Find the vector $\tilde{y}_m$ which solves the linear system $\tilde{H}_m\tilde{y} = \tilde{\beta}e_1$.
    • Compute $\delta^{(m)} = \delta^{(0)} + P^{-1}D_u^{-1}\tilde{z}^{(m)}$, where $\tilde{z}^{(m)} = \tilde{V}_m\tilde{y}_m$.
    **GMRES:** Define $\tilde{\tilde{H}}_m$ to be the $(m+1) \times m$ (Hessenberg) matrix whose
    nonzero entries are the coefficients $\tilde{h}_{ij}$, $1 \le i \le j+1$, $1 \le j \le m$ and
    $\tilde{V}_m \equiv [\tilde{v}_1, \cdots, \tilde{v}_m]$.
```

- Find the vector $\tilde{y}_m$ which minimizes $\|\tilde{\beta}e_1 - \tilde{\tilde{H}}_m\tilde{y}\|_2$ over all vectors $\tilde{y}$ in $\mathbf{R}^m$.
- Compute $\delta^{(m)} = \delta^{(0)} + P^{-1}D_u^{-1}\tilde{z}^{(m)}$, where $\tilde{z}^{(m)} = \tilde{V}_m\tilde{y}_m$.

In this formulation, we have elected to precondition first and scale second. Alternatively, one could do these in the opposite order. The parentheses in the calculation of $\tilde{J}\tilde{v}_j$ indicate that this product is found by first forming $D_u^{-1}\tilde{v}_j$, then solving the system $Pw = D_u^{-1}\tilde{v}_j$, multiplying w by J, and then finally multiplying the result by D_F. Recall that the matrix J here is never formed, and so each operation must be done separately.

6. Numerical testing.

In this section, we discuss the relevant implementation details of the methods developed in the previous sections, and then present some results of testing the methods on several problems. We have implemented both global strategies in one solver, the format of which was based upon the overall design of such a package presented in Dennis and Schnabel [7].

6.1. An experimental solver.

A solver package called NKSOL (Nonlinear Krylov SOLver for nonlinear systems of equations) has been developed which implements the above methods. NKSOL allows the user to select from among three basic options:

- Arnoldi/IOM as the linear Krylov method with the linesearch backtracking global strategy,
- GMRES/IGMRES as the linear Krylov method with the linesearch backtracking global strategy, and
- GMRES as the linear Krylov method with the dogleg global strategy.

The driver routine, NKSOL, checks for valid input, handles the initial startup of the iteration, and calls a routine NKSTOP to decide when to stop the iteration. It also calls routines MODEL, LNSRCH, and DOGDRV, which perform the following functions. MODEL calls SLVS, which in turn calls either SPIOM or SPIGMR to solve the Newton equations approximately using Arnoldi or GMRES, respectively. LNSRCH determines an acceptable step in the direction provided by MODEL using the linesearch backtracking strategy of §3. DOGDRV is the dogleg strategy driver. It calls DOGSTP, which computes the point on the dogleg curve corresponding to the current trust region size, and TRGUPD, which determines if the step provided by DOGSTP is acceptable and adjusts the trust region size accordingly. See Fig. 6.1.

The user must supply a routine for calculating $F(u)$, and may optionally supply any or all of three additional routines. By default, finite-differences are used to calculate Jv. However, the user may also supply a routine JAC to perform this multiplication. The other user-supplied routines are PSET and PSOL which involve the preconditioner matrix P. Routine PSOL solves the linear system $Pw = c$, and PSET is called once per Newton iterate to set up any matrix data associated with P.

One detail of particular importance is the selection of σ in the difference quotient approximation (2.11). When using finite differences to form an approximate Jacobian matrix in the context of Newton's method, Dennis and Schnabel [7] suggest a stepsize of the form

$$h_j = \sqrt{\eta}\max\{|u_j|, \mathrm{typ}u_j\}\ \mathrm{sign}(u_j), \tag{6.1}$$

in the difference quotient

$$\frac{F(u + h_je_j) - F(u)}{h_j}.$$

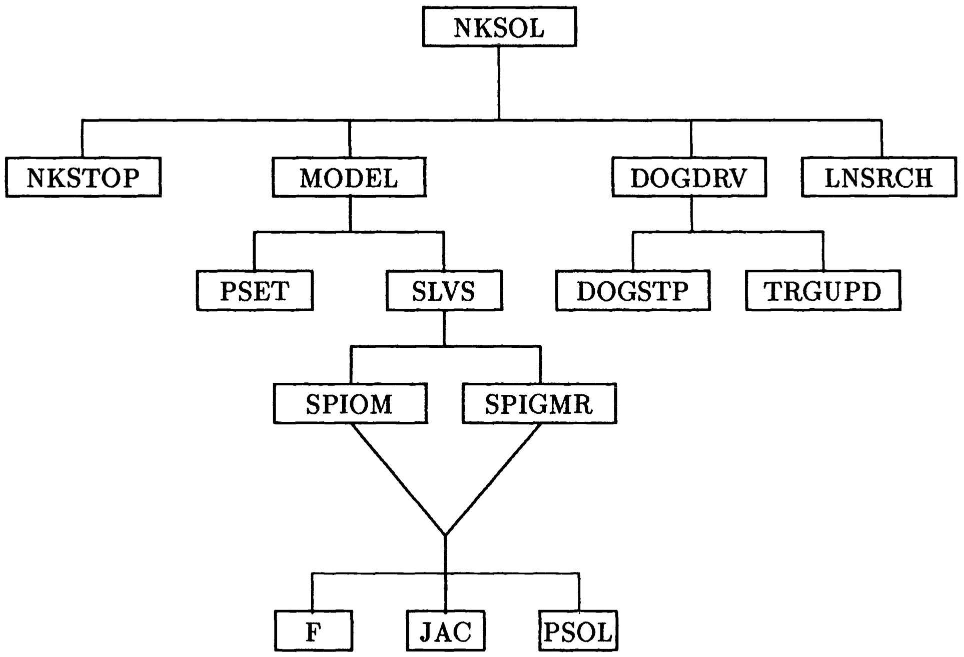


FIG. 6.1. *Simplified Block Structure of NKSOL.*

Here, η represents the relative error in computing $F(u)$, e_j is the jth standard basis vector in $\mathbf{R}^N$, and $\text{typ}u_j > 0$ is a typical size of u_j provided by the user. The size of η depends upon how much work is needed to calculate $F(u)$. In the absence of any information, a value of η equal to machine epsilon is appropriate. The above formula gives an approximation for the jth column of the Jacobian matrix $J(u)$.

In our setting we need only approximate the operation Jv, for a given vector v, and not J itself. To modify (6.1), first note that it can be rewritten as

$$h_j = \sqrt{\eta}\max\{|u^T e_j|, \text{typ}u^T e_j\}\,\text{sign}(u^T e_j),$$

where $\text{typ}u = [\text{typ}u_1, \cdots, \text{typ}u_N]^T$. Note also that $\|e_j\|_2 = 1$. If we ignore scaling for the moment, and for a given vector v let $w = v/\|v\|_2$, then an analogous choice for $\bar{\sigma}$ in the difference quotient

$$Jw \approx \frac{F(u+\bar{\sigma}w) - F(u)}{\bar{\sigma}}$$

is

$$\begin{aligned}\bar{\sigma} &= \sqrt{\eta}\max\{|u^T w|, \text{typ}u^T|w|\}\,\text{sign}(u^T w)\\ &= \frac{\sqrt{\eta}}{\|v\|_2}\max\{|u^T v|, \text{typ}u^T|v|\}\,\text{sign}(u^T v),\end{aligned}$$

where $|w| = [|w_1|, \cdots, |w_N|]^T$, with a similar definition for $|v|$. Choosing $\sigma = \bar{\sigma}/\|v\|_2$ gives us an appropriate value to use in (2.11). If we include scaling, then $\|\cdot\|_2$ and its associated dot product should be replaced by the scaled norm $\|D_u\cdot\|_2$ and its dot product $(D_u\cdot, D_u\cdot)$. Hence, for a given v, letting $w = v/\|D_u v\|_2$ leads to

$$\begin{aligned}\bar{\sigma} &= \sqrt{\eta}\max\{|(D_u u)^T D_u w|, (D_u \mathrm{typ}u)^T D_u |w|\}\ \mathrm{sign}((D_u u)^T D_u w) \\ &= \frac{\sqrt{\eta}}{\|D_u v\|_2}\max\{|(D_u u)^T D_u v|, (D_u \mathrm{typ}u)^T D_u |v|\}\ \mathrm{sign}((D_u u)^T D_u v),\end{aligned}$$

and $\sigma = \bar{\sigma}/\|D_u v\|_2$.

The entries of the diagonal scaling matrix D_u are chosen so as to make the components of the scaled vector $\tilde{u} = D_u u$ all roughly equal in magnitude. Hence, if $D_u = \mathrm{diag}\{d_{11}, \cdots, d_{NN}\}$, then one should use $d_{jj} = \mathrm{typ}u_j^{-1}$ for each j. A similar choice for the entries of the D_F matrix can be made.

The stopping criteria used are those described in Chapter 7 of [7], and are repeated here for convenience. The first test determines whether u_n solves the problem (1.1), i.e., whether $F(u_n) \approx 0$. This is accomplished by using

$$\|D_F F(u_n)\|_\infty \le \mathrm{FTOL}, \tag{6.2}$$

where a typical value for FTOL is around 10^{-5}. The next test determines whether the algorithm has converged or stalled at u_n. It is of the form

$$\|\mathrm{rel}u\|_\infty \le \mathrm{STPTOL}, \tag{6.3}$$

where rel$u$ is a vector of length N, which measures the relative change in u from one step to the next. Its components are defined by

$$\mathrm{rel}u_j = \frac{|(u_{n+1})_j - (u_n)_j|}{\max\{|(u_{n+1})_j|, \mathrm{typ}u_j\}}$$

for $j = 1, \cdots, N$. If t significant digits are desired in the solution, then STPTOL should be set to 10^{-t}. (Alternatively, the user can supply his own routines for calculating the size of $F(u_n)$ and the relative change in u that appears in (6.2) and (6.3).) A maximum steplength STPMX is imposed in both the linesearch and dogleg strategies, and if five consecutive steps are this long, then the algorithm is terminated. There is also a maximum number ITMAX of iterations that can be performed on any one call to NKSOL. Currently, this value is 200, but it can optionally be set by the user.

In the test results of the remainder of this section, the following counters and a return flag will be helpful. These are defined as follows:

NFE	–	number of F evaluations
NNI	–	number of nonlinear iterations
NLI	–	number of linear iterations within the Krylov method
NB	–	number of backtracks within the linesearch algorithm, and number of extra F evaluations used by the dogleg strategy
NCFL	–	number of times that the linear solver failed to reduce the residual norm by a factor η_n in $m_{\max}$ steps
ITERM	–	termination flag (=1 if (6.2) holds, =2 if (6.3) holds, and =3 if (3.10) fails to hold).

For ϵ_n we use $\epsilon_n = \eta_n\|F\|_2$, where $\eta_n = (\frac{1}{2})^n$ for $n = 1, 2, \cdots$, and $F = F(u_n)$. A default value of $m_{\max} = 10$ is used with the full orthogonalization version of each Krylov method. If $m = m_{\max}$, but $\|r^{(m)}\|_2 > \eta_n\|F\|_2$, we simply go ahead and use the last computed GMRES or Arnoldi step. The counter NCFL equals the number of Newton iteration steps for which this happened.

TABLE 6.1
Results for the Bratu problem, $\lambda = 1$.

	No Preconditioning			Laplacian Preconditioning		
	MF = 1	MF = 2	MF = 3	MF =1	MF = 2	MF = 3
NFE	151	205	150	28	28	27
NNI	15	20	15	6	6	6
NLI	134	184	134	20	21	20
NCFL	13	18	13	0	0	0
NB	1	0	0	1	0	0
ITERM	1	1	1	1	1	1

6.2. Test problem 1. Solution of a Bratu problem. As a first test problem we chose to solve the nonlinear partial differential equation

$$-\Delta u + \alpha u_x + \lambda e^u = f \tag{6.4}$$

over the unit square of $\mathbf{R}^2$ with Dirichlet boundary conditions. This is a standard problem, a simplified form of which is known as the Bratu problem [13]. After discretization by 5-point finite differencing, we obtain a large system of nonlinear equations of size N, where $N = {n_x}^2$ and n_x is the number of meshpoints in each direction. The right-hand side f is chosen so that the solution of the discretized problem is known to be the constant unity. As a result, the equation will always have at least one solution. In fact, it is known that for $\lambda \geq 0$ there is always a unique solution to the problem (see [13]). In this test we took $n_x = 32$, yielding a nonlinear system of $N = 1024$ unknowns, and $\alpha = 10.0, \lambda = 1.0$. It is possible to precondition this problem by the Laplacian operator. To this end we can use a fast Poisson solver such as the subroutine HWSCRT from FISHPACK [26]. We tested the three basic methods from NKSOL, each of them with and then without this preconditioning. The three methods correspond to the following method flags:

MF = 1 –	The dogleg technique using GMRES as a linear solver.
MF = 2 –	The backtracking linesearch technique using Arnoldi's method as a linear solver.
MF = 3 –	The backtracking linesearch technique using the GMRES method as a linear solver.

For Arnoldi and GMRES the default value of $m_{\max} = 10$ was used, and the values of FTOL and STPTOL were 10^{-7} and 10^{-10}, respectively. No scaling was used and the initial guess was always taken to be zero. Table 6.1 shows the results for all three methods with and without preconditioning, while the graph of Fig. 6.2 plots the value of $f = \frac{1}{2}F^TF$ on a logarithmic scale in powers of 10 as a function of NFE, the number of function evaluations consumed to reach that value of f. Here, the performances of the three basic methods MF=1,2,3 differ very little as is shown in Table 6.1 and in the plot of Fig. 6.2, with the exception that MF=1 and 3 have a slight edge over MF=2. On the other hand, preconditioning was extremely helpful. Note that both the number of nonlinear iterations and the total number of linear steps is much reduced by the use of the preconditioner. We should mention that for this problem it is possible to take advantage of other subroutines from FISHPACK. For example, instead of preconditioning with the Laplacian we could have used BLCKTRI to precondition by the discrete version of the partial differential operator $\Delta + \partial/\partial x$ or even of $\Delta + \kappa + \partial/\partial x$ where κ could be, for example, some average of $\lambda e^{u_{n-1}}$ over the domain.

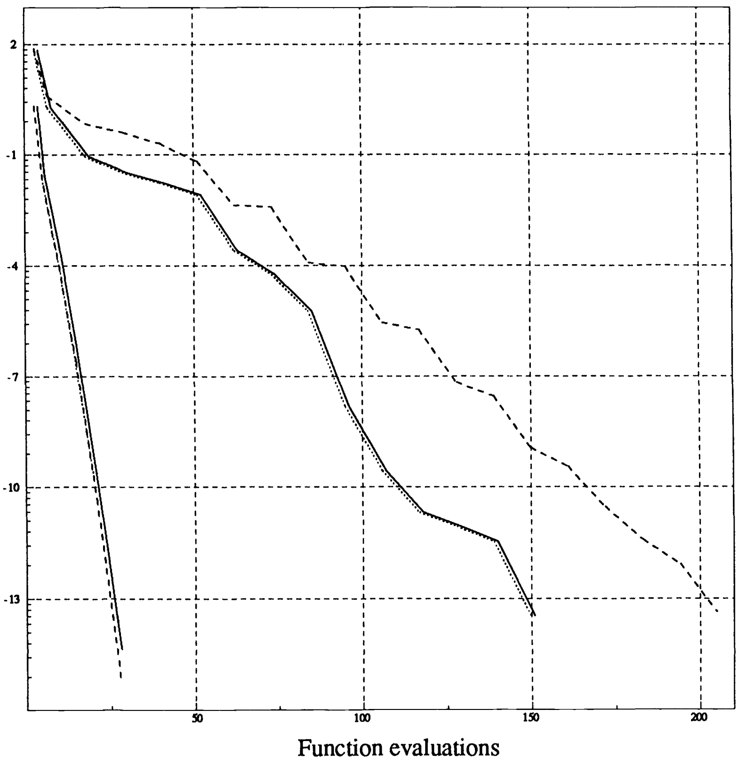


FIG. 6.2. *Results for the Bratu problem, $\lambda = 1$. Solid line: $MF = 1$; dashed line: $MF = 2$; dotted line: $MF = 3$. Lower curves: with preconditioning; upper curves: no preconditioning.*

In our second experiment with this test problem, we took a negative value for λ to make the problem more difficult to solve. When $\lambda < 0$ the Jacobians in Newton's method may not be positive real and this is a source of difficulty for the linear system solvers. With $\lambda = -5$ we obtain the results of Table 6.2 and the plot of Fig. 6.3. We observe again that there is little difference between the three methods, again with the exception that MF=1 and 3 appear to have a slight edge over MF=2 when there is no preconditioning.

6.3. Test problem 2. The driven cavity problem. This second test problem is the classical *driven cavity* problem from incompressible fluid flow. In stream function–vorticity formulation the equations are

$$\nu\Delta\omega + (\psi_{x_2}\omega_{x_1} - \psi_{x_1}\omega_{x_2}) = 0 \text{ in } \Omega, \tag{6.5}$$

$$-\Delta\psi = \omega \text{ in } \Omega, \tag{6.6}$$

$$\psi = 0 \text{ on } \partial\Omega, \tag{6.7}$$

$$\frac{\partial\psi}{\partial n}(x_1, x_2)|_{\partial\Omega} = \begin{cases} 1 & \text{if } x_2 = 1, \\ 0 & \text{if } 0 \le x_2 < 1. \end{cases} \tag{6.8}$$

Here, $\Omega = \{(x_1, x_2) : 0 < x_1 < 1, 0 < x_2 < 1\}$, and the viscosity ν is the reciprocal of the Reynolds number *Re*. In terms of ψ alone, (6.5) and (6.6) are replaced by

$$\nu\Delta^2\psi + (\psi_{x_2}(\Delta\psi)_{x_1} - \psi_{x_1}(\Delta\psi)_{x_2}) = 0 \text{ in } \Omega, \tag{6.9}$$

subject to the boundary conditions (6.7) and (6.8). For more details on the problem definition see [13].

TABLE 6.2
Results for the Bratu problem, $\lambda = -5$.

	No Preconditioning			Laplacian Preconditioning		
	MF = 1	MF = 2	MF = 3	MF =1	MF = 2	MF = 3
NFE	195	230	216	30	29	29
NNI	19	22	21	6	6	6
NLI	174	204	194	22	22	22
NCFL	17	20	19	0	0	0
NB	1	3	0	1	0	0
ITERM	1	1	1	1	1	1

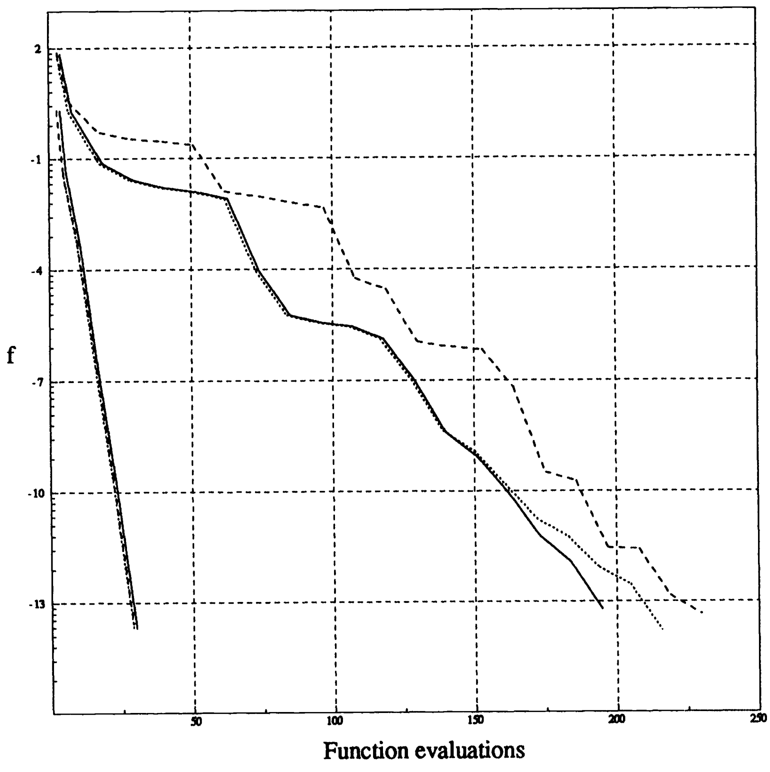


FIG. 6.3. *Results for the Bratu problem, $\lambda = -5$. Solid line: $MF = 1$; dashed line: $MF = 2$; dotted line: $MF = 3$. Lower curves: with preconditioning; upper curves: no preconditioning.*

TABLE 6.3
Results for the driven cavity problem (Re = 500).

BIHAR	Cholesky Decomposition		
	MF = 1	MF = 2	MF = 3
NFE	146	308	145
NNI	14	27	14
NLI	130	261	130
NCFL	11	26	11
NB	1	19	0
ITERM	1	1	1

TABLE 6.4
Results for the driven cavity problem (Re = 1500 *and* 2000).

	Re = 1500			Re = 2000	
	MF = 1	MF = 2	MF = 3	MF = 1	MF = 3
NFE	484	393	472	552	567
NNI	31	16	30	35	36
NLI	451	226	436	512	527
NCFL	30	15	29	34	35
NB	1	150	5	4	3
ITERM	1	3	1	1	1

Equation (6.9) was discretized using piecewise linear finite elements in a manner similar to that for the biharmonic problem $\Delta^2\psi = f$ outlined in §3 of [14]. This discretization is also equivalent to that obtained using standard finite differences. The resulting nonlinear system has $N = 63^2 = 3969$ unknowns. For a preconditioner, we use the discretized version of $P = \nu\Delta^2$. For a uniform mesh, systems of the form $Pw = c$ can be solved very efficiently using a fast solver, for example BIHAR developed by Bjørstad [1]. This preconditioner should be very effective for small Reynolds numbers, with decreasing effectiveness as the Reynolds number grows. For comparison, we considered Re = 500, 1500, 2000, 3000, and 5000. All runs were made on a CRAY-1 at LLNL, with FTOL = 10^{-7}, STPTOL = 10^{-8}, and the scaling matrices D_F and D_u both equal to the identity (i.e., no scaling). The initial guess ψ_0 was simply the zero vector.

The results for Re = 500 are given in Table 6.3. We used the Cholesky decomposition option in BIHAR for the solution of the biharmonic problem. This option computes an exact solution with a minimal storage requirement. For this Reynolds number, the preconditioner is very effective and the two GMRES method options perform similarly, whereas the Arnoldi/Linesearch option takes more nonlinear iterations but still achieves convergence. See Fig. 6.4 for a plot of function evaluations versus values of $f = \frac{1}{2}F^TF$, and Fig. 6.5 for a contour plot of the stream function values in this case. The contour levels plotted are

$$\begin{aligned}\psi &= -0.1, -0.08, -0.06, -0.04, -0.02, 0.0,\\ &\quad 0.0025, 0.001, 0.0005, 0.0001, 0.00005.\end{aligned}$$

The results for Re = 1500 and 2000 are given in Table 6.4. For these higher Reynolds numbers the preconditioner is less effective, and the original runs (using

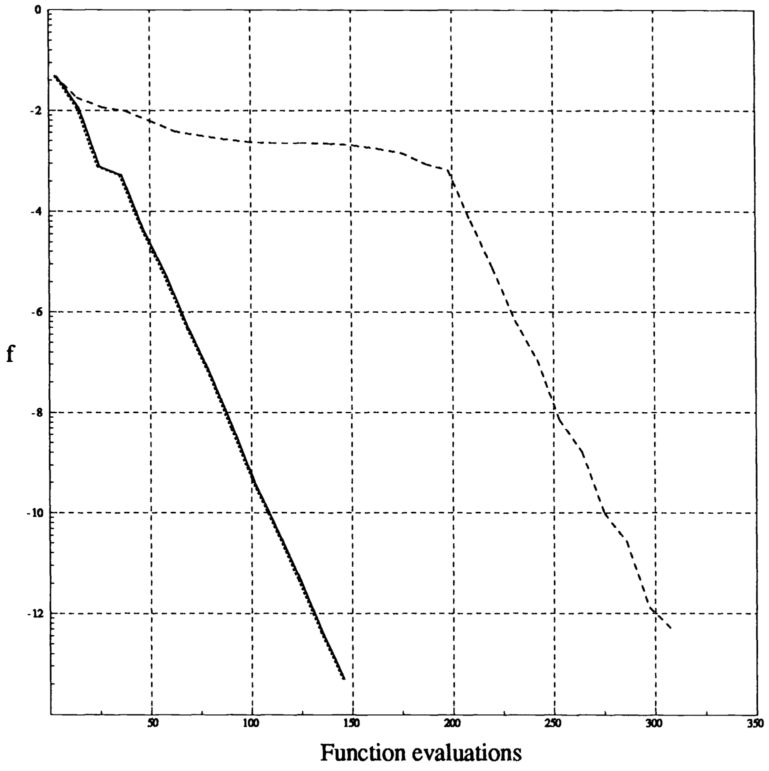


FIG. 6.4. *Reynolds number* 500. *Solid line*: $MF = 1$; *dashed line*: $MF = 2$; *dotted line*: $MF = 3$. *Cholesky preconditioning.*

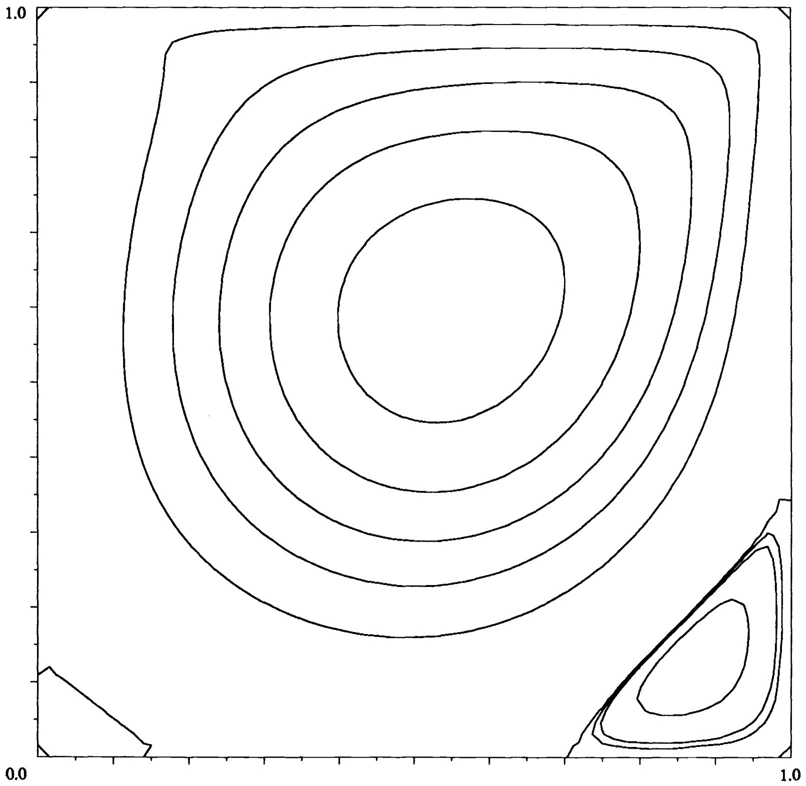


FIG. 6.5. *Reynolds number* 500.

TABLE 6.5
Results for the driven cavity problem (*Re* = 3000 *and* 5000).

Re	3000	5000
	MF = 1	MF = 1
NFE	904	1407
NNI	57	88
NLI	845	1315
NCFL	56	87
NB	1	3
ITERM	1	1

the default value of $m_{\max} = 10$) failed to converge for all MF values. Increasing $m_{\max}$ to 15 yielded the results in the table. A possible reason for the convergence failures is that since the preconditioner is less effective, the default Krylov subspace dimension is not large enough to generate sufficiently good descent directions for the nonlinear iteration. In addition, the Arnoldi/Linesearch combination performed very poorly, and the high number of backtracks suggest that Arnoldi is producing very poor descent directions (even with the higher $m_{\max}$ value). We say more about this below. See Figs. 6.6 and 6.7 for function evaluations versus f, and Figs. 6.8 and 6.9 for contour plots. Table 6.5 contains the results for Reynolds numbers 3000 and 5000. We elected to test only the dogleg strategy for these higher Reynolds numbers. While GMRES is having difficulty solving the linear systems for high Reynolds numbers the nonlinear iteration is still achieving convergence. (See Figs. 6.10–6.13.)

Finally, we note that the Arnoldi/Linesearch option in general performs very poorly for the large Reynolds number cases. This may be partly due to the fact that the Jacobian matrix $J(\psi)$ becomes nearly skew-symmetric for small ν. While it is not surprising that both Arnoldi and GMRES may perform poorly in this case, the directions each computes can be vastly different. It follows from (3.3) and (3.4) that the derivative $\nabla f^T \delta$ is independent of the residual associated with the Arnoldi step $\delta = \delta_A$, whereas this is not the case for the GMRES step $\delta = \delta_{GM}$. If we use the norm of the residual as a measure of how good an approximation δ is to the full Newton step, then it appears that the GMRES step δ_{GM} may in general be preferable. In other words, a poor GMRES step is in general better than a poor Arnoldi step.

7. Conclusion. Krylov subspace methods for linear systems can be combined with Newton iteration and globally convergent strategies to obtain effective algorithms for general nonlinear systems of equations. We have shown how the Arnoldi and GMRES methods can be fitted into such a combination and have discussed some of the relevant implementation details. We have also implemented these algorithms in an experimental solver NKSOL. The test problems we have considered show that these methods (with suitable preconditioning) can be quite effective for some classes of problems. Propositions 3.1 and 3.2 and the remarks at the end of the previous section indicate that GMRES may be slightly better than other available Krylov methods, at least in the nonlinear equations setting. Similar to the situation in linear equations, one observes that the choice of a preconditioner is more important than that of the Krylov subspace method. In many instances, nonlinear preconditioners can be built from linear parts of the nonlinear equations as was shown in the examples of § 6. Another way of preconditioning is to transform the initial nonlinear equation, for example, by solving $u - M(u) = 0$ where the operator comes from some known fixed

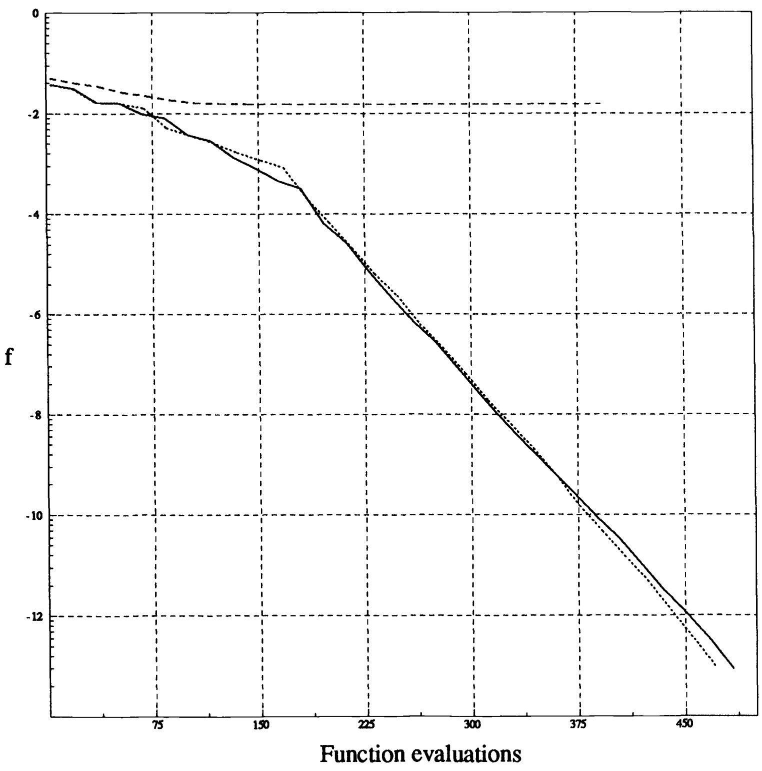


FIG. 6.6. *Reynolds number* 1500. *Solid line*: $MF = 1$; *dashed line*: $MF = 2$; *dotted line*: $MF = 3$. *Cholesky preconditioning.*

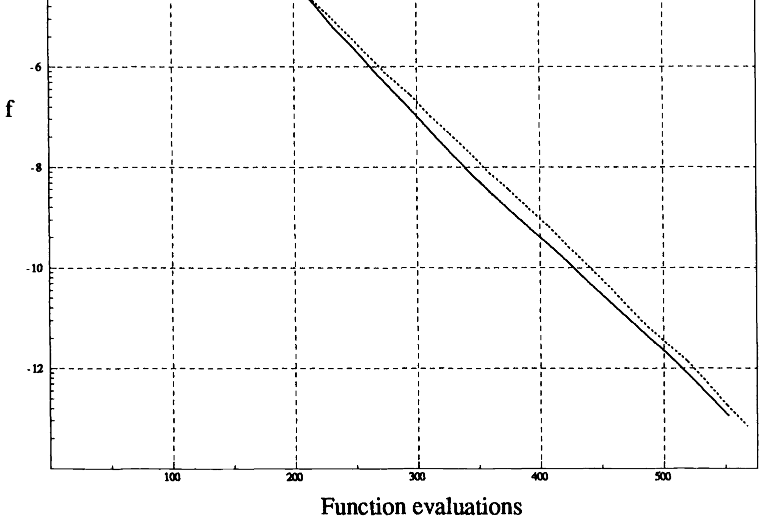


FIG. 6.7. *Reynolds number* 2000. *Solid line*: $MF = 1$; *dotted line*: $MF = 3$. *Cholesky preconditioning.*

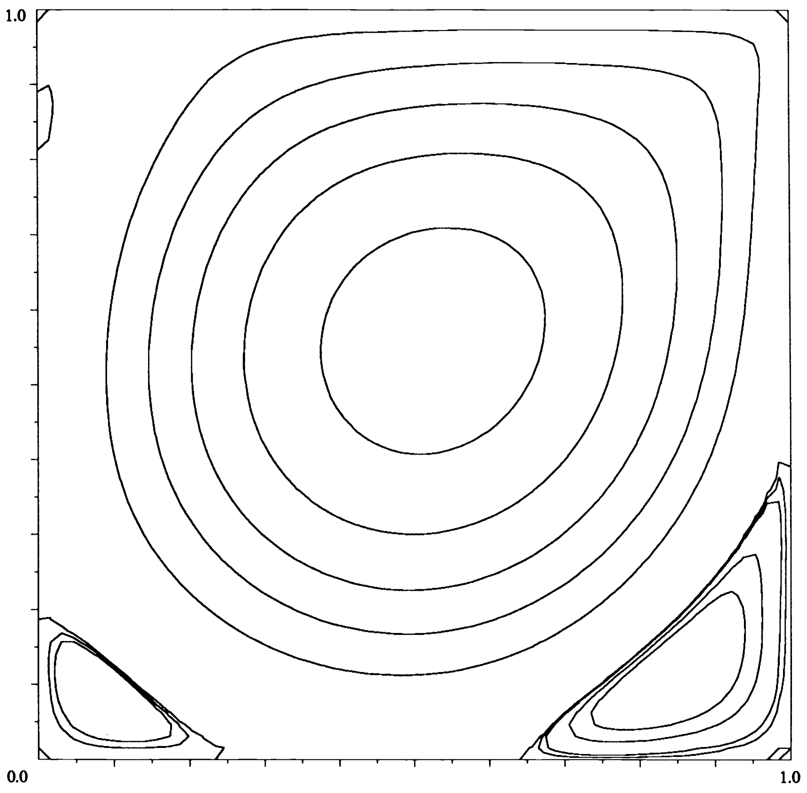


FIG. 6.8. *Reynolds number* 1500.

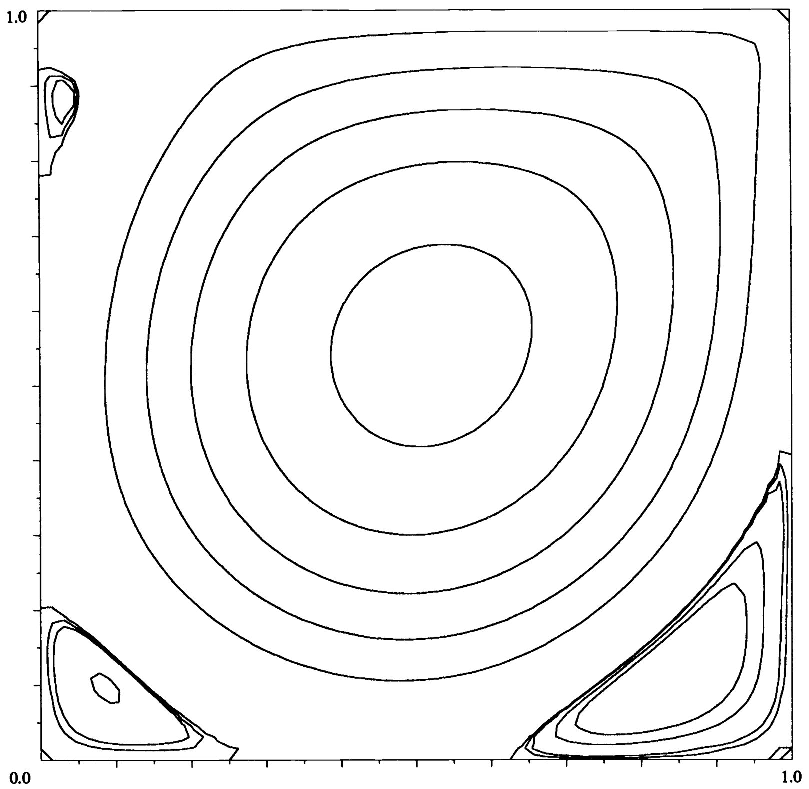


FIG. 6.9. *Reynolds number* 2000.

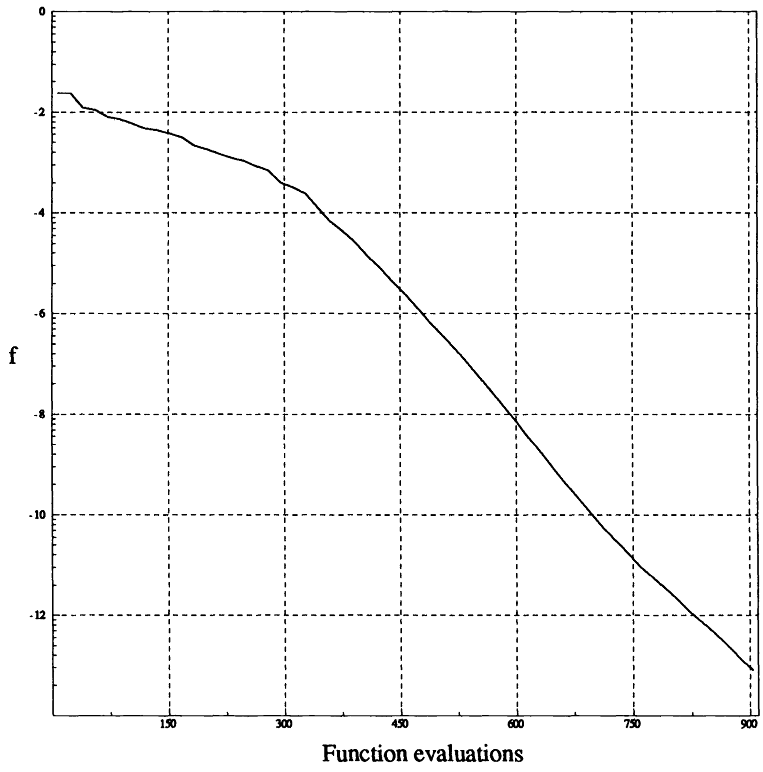


FIG. 6.10. *Reynolds number* 3000. *Solid line*: $MF = 1$. *Cholesky preconditioning.*

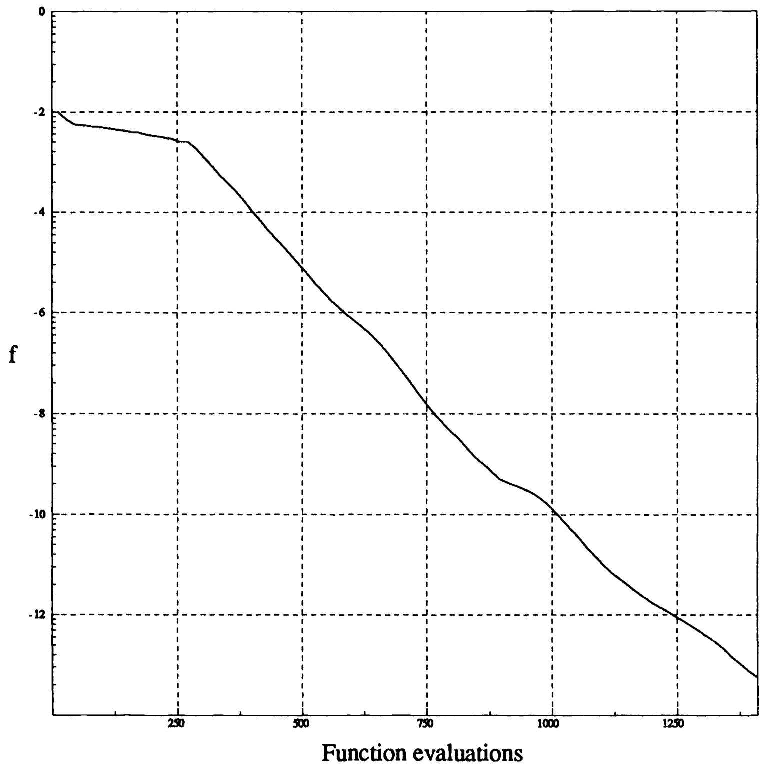


FIG. 6.11. *Reynolds number* 5000. *Solid line*: $MF = 1$. *Cholesky preconditioning.*

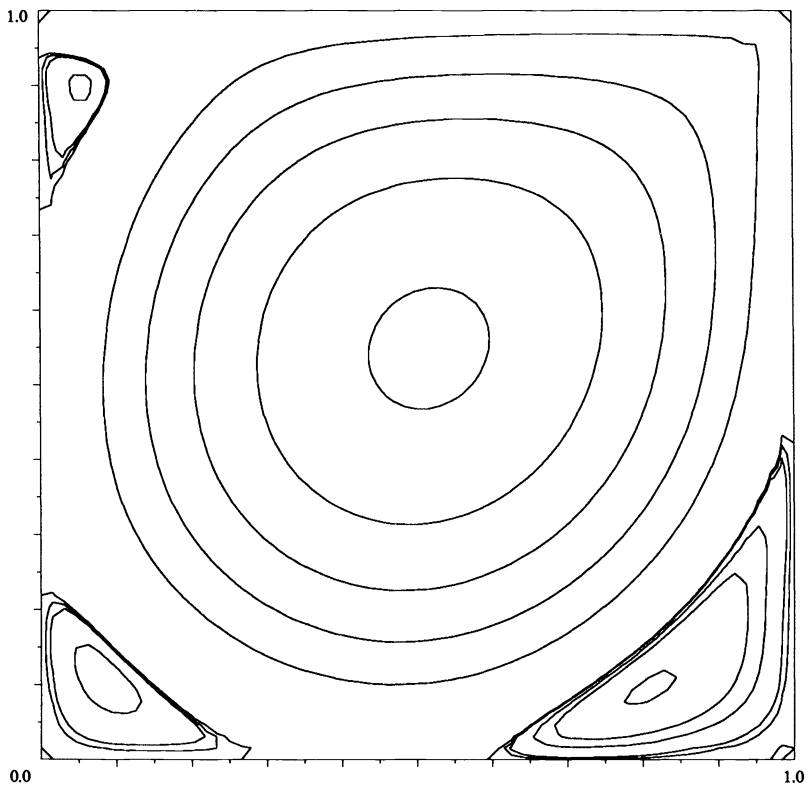


FIG. 6.12. *Reynolds number* 3000.

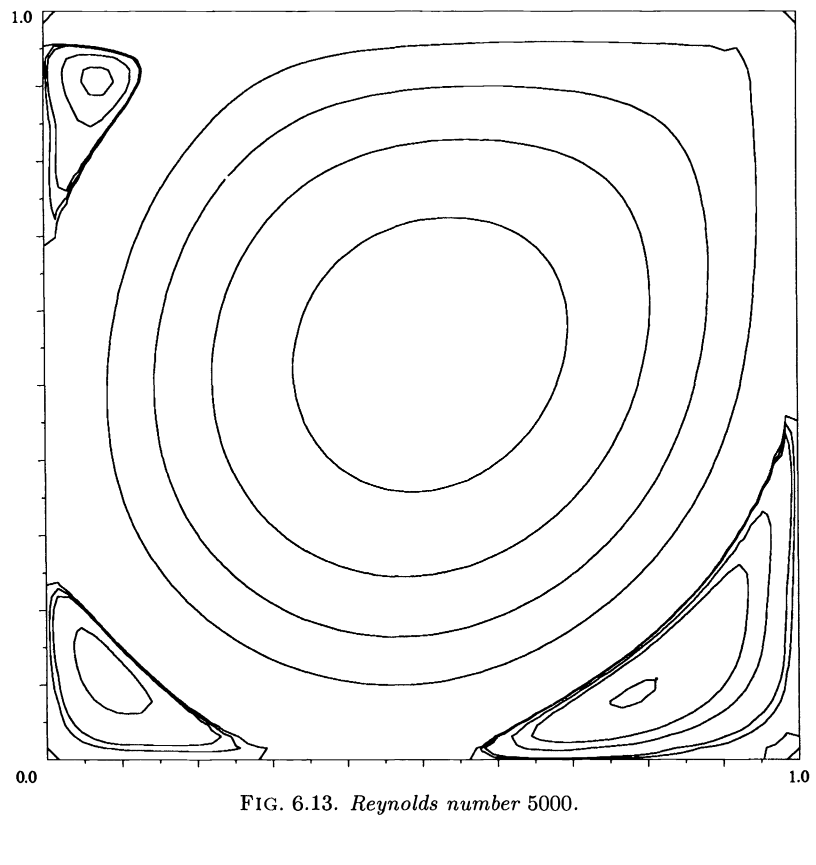


FIG. 6.13. *Reynolds number* 5000.

point iteration $u_{n+1} = M(u_n)$. This opens up many possibilities for using Krylov subspace methods as accelerators of existing codes based on nonlinear stationary iterations. There are many issues that remain to be investigated. Among them is the possibility of exploiting information gathered during the previous nonlinear steps, such as previous Krylov subspaces and the corresponding Jacobian in that space. Another interesting question, related to the fluid dynamics problem of § 6.2, is whether solving the stationary problem directly is more effective than solving the time-dependent problem over a large enough time period. This second approach would in effect constitute a sort of continuation technique with continuation parameter the time t, and it might very well be the case that for very hard problems a combination of the methods described in this paper and a form of continuation will be the most robust approach.

Finally, this paper has focused on presenting a general purpose nonlinear equation solver based on nonlinear Krylov subspace techniques, enhanced by global convergence strategies. The algorithms we have implemented are all based on extrapolations of globally convergent methods for full N-dimensional quadratic models. In future work, we intend to study the convergence properties of these combined global/Newton–Krylov methods.

REFERENCES

[1] P. BJØRSTAD, *Fast numerical solution of the biharmonic Dirichlet problem on rectangles,* SIAM J. Numer. Anal., 20(1983), pp. 59-71.

[2] P. N. BROWN, *A local convergence theory for combined inexact-Newton/finite-difference projection methods,* SIAM J. Numer. Anal., 24(1987), pp. 407-434.

[3] P. N. BROWN AND A. C. HINDMARSH, *Matrix-free methods for stiff systems of* ODEs, SIAM J. Numer. Anal., 24(1987), pp. 610-638.

[4] ________, *Reduced storage methods in stiff* ODE *systems,* J. Appl. Math. Comput., 31(1989), pp. 40-91.

[5] 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. Comput., 7(1986), pp. 378-417.

[6] R. S. DEMBO, S. C. EISENSTAT, AND T. STEIHAUG, *Inexact Newton methods,* SIAM J. Numer. Anal., 19(1982), pp. 400-408.

[7] J. E. DENNIS AND R. B. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations,* Prentice-Hall, Englewood Cliffs, NJ, 1983.

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

[9] L. E. ERIKSSON AND A. RIZZI, *Analysis by computer of the convergence of discrete approximations to the Euler equations,* in Proc. 1983 AIAA Conference, Denver, 1983; AIAA paper number 83-1951, Denver, CO, 1983, pp. 407-442.

[10] D. GAY, *Computing optimal locally constrained steps,* SIAM J. Sci. Statist. Comput., 2(1981), pp. 186-197.

[11] W. C. GEAR AND Y. SAAD, *Iterative solution of linear equations in* ODE *codes,* SIAM J. Sci. Statist. Comput., 4(1983), pp. 583-601.

[12] P. E. GILL, W. MURRAY, AND M. WRIGHT, *Practical Optimization,* Academic Press, New York, 1981.

[13] R. GLOWINSKI, H. B. KELLER, AND L. REINHART, *Continuation-conjugate gradient methods for the least squares solution of nonlinear boundary value problems,* SIAM J. Sci. Statist. Comput., 6(1985), pp. 793-832.

[14] R. GLOWINSKI AND O. PIRONNEAU, *Numerical methods for the first biharmonic equation and for the two-dimensional Stokes problem,* SIAM Rev., 21(1979), pp. 167-212.

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

[16] T. KERKHOVEN AND Y. SAAD, *Acceleration techniques for decoupling algorithms in semiconductor simulation,* CSRD Report 684, University of Illinois at Urbana Champaign, Urbana, IL, 1987.

[17] W. L. MIRANKER AND I.-L. CHERN, *Dichotomy and conjugate gradients in the stiff initial value problem,* Linear Algebra Appl., 36(1981), pp. 57-77.

[18] J. J. MORÉ, *The Levenberg–Marquardt algorithm: Implementation and theory*, in Numerical Analysis, G. A. Watson, ed., Lecture Notes in Mathematics 630, Springer-Verlag, Berlin, New York, 1977, pp. 105-116.

[19] J. J. MORÉ, B. S. GARBOW, AND K. E. HILLSTROM, *User guide for* MINPACK-1, Tech. Report ANL-80-74, Argonne National Laboratories, Argonne, IL, 1980.

[20] D. P. O'LEARY, *A discrete Newton algorithm for minimizing a function of many variables,* Math. Programming, 23 (1982), pp. 20-33.

[21] M. J. D. POWELL, *A hybrid method for nonlinear equations*, P. Rabinowitz, ed., Numerical Methods for Nonlinear Equations, Gordon-Breach, New York, 1970.

[22] Y. SAAD, *Krylov subspace methods for solving unsymmetric linear systems,* Math. Comp., 37(1981), pp. 105-126.

[23] Y. SAAD AND M. H. SCHULTZ, *Conjugate gradient-like algorithms for solving nonsymmetric linear systems,* Math. Comp., 44(1985), pp. 417-424.

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

[25] T. STEIHAUG, *The conjugate gradient method and trust regions in large scale optimization,* SIAM J. Numer. Anal., 20 (1983), pp. 626-637.

[26] P. N. SWARTZTRAUBER AND R. A. SWEET, *Algorithm* 541: *Efficient Fortran subprograms for the solution of separable elliptic partial differential equations,* ACM Trans. Math. Software, 5(1979), pp. 352-364.

[27] L. B. WIGTON, D. P. YU, AND N. J. YOUNG, GMRES *acceleration of computational fluid dynamics codes*, in Proc. 1985 AIAA Conference, Denver, CO, 1985.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 3, pp. 482-502, May 1990

SMOOTHING PERIODIC CURVES BY A METHOD OF REGULARIZATION*

CHRISTINE THOMAS-AGNAN†

Abstract. The present work develops an extension of the theory of polynomial smoothing splines for the statistical problem of estimating a periodic curve based on discrete and noisy observations of the curve itself or of a convolution functional of the curve. These estimates, called periodic α-splines, are still of the method of regularization type, with a penalty involving Fourier coefficients. The questions of asymptotic rate of convergence of the integrated mean square error and the relationships with kernel estimates in the case of equispaced design points are addressed.

Key words. smoothing splines, method of regularization, kernels, nonparametric regression, Fourier series

AMS(MOS) subject classifications. primary 62G05; secondary 62G20, 65D07

1. Introduction.

The problem of estimating a periodic curve given incomplete and noisy observations arises naturally in a variety of experimental situations, including, for example, the estimation of log-spectral density in time-series analysis (see Wahba and Wold [12]). We consider the following model. Let $f(\cdot)$ be an unknown function, from $\mathbb{R}$ to $\mathbb{R}$, periodic with period 1, and let the observations $(y_1, y_2, \cdots, y_n) \in \mathbb{R}^n$ be related to $f(\cdot)$ by

$$y_i = L_i f + \varepsilon_i, \qquad i = 1, \cdots, n \tag{1}$$

where the errors ε_i are realizations of n random variables with mean zero and covariances $E(\varepsilon_i \varepsilon_j) = \sigma^2 \delta_{ij}$, and L_i, $i = 1, \cdots, n$, are n continuous linear functionals on a topological vector space of functions $\mathscr{C}$ to which $f(\cdot)$ is assumed to belong.

We will first specialize to the evaluation case when the functionals L_i are merely observations of the function $f(\cdot)$ at n design points $t_1, \cdots, t_n$, which we may assume to lie in $[0, 1]$ without loss of generality:

$$L_i f = f(t_i), \qquad i = 1, \cdots, n. \tag{2}$$

A least squares estimate is often used when the space $\mathscr{C}$ contains elements that can be entirely determined by a finite number of real parameters ("parametric" method) and is obtained by minimizing

$$\mathrm{LS}(g) = \frac{1}{n} \sum_{i=1}^{n} (g(t_i) - y_i)^2 \tag{3}$$

over $g(\cdot) \in \mathscr{C}$.

When we do not want to make such a strong assumption on $\mathscr{C}$ a natural generalization is the penalized least square estimate: to avoid simple interpolation of the data, we add a penalty term to LS penalizing highly irregular or physically less plausible curves. This is the case for periodic smoothing splines where we use for $\mathscr{C}$ the periodic Sobolev spaces: $W_m^{\mathrm{per}}(0, 1)$ of periodic real-valued functions with absolutely continuous kth derivative, $0 \leq k \leq m-1$, and square integrable mth derivative, and the penalty

$$J(g) = \lambda \int_0^1 |g^{(m)}(t)|^2 \, dt \tag{4}$$

* Received by the editors December 30, 1987; accepted for publication (in revised form) June 20, 1989.

† Laboratoire de Statistique et Probabilités, UA-CNRS 745, Université Paul Sabatier, 118, route de Narbonne, 31062 Toulouse Cedex, France. Present address, 7 rue des Saules, 31400 Toulouse, France.

where λ determines the balance between fit and smoothness (see Wahba [13]). The fitting of a function $f(\cdot)$ to discrete data y_i by minimization of a functional such as $\mathrm{LS}(g)+J(g)$ is also called the method of regularization.

Our approach is to look at such a method with a penalty based on the Fourier coefficients of the function $g(\cdot)$, which are defined for $g(\cdot)\in L_2(0,1)$ by

$$g_k=\int_0^1 g(u)\,e^{-2\pi iku}\,du\in\mathbb{C},\qquad k\in\mathbb{Z}. \tag{5}$$

The motivation for this is that the smoothness of a periodic function $g(\cdot)$ is determined by the asymptotic behaviour of the sequence g_k. For example, a function $g(\cdot)$ with derivatives $g^{(q)}(\cdot)$ absolutely integrable and absolutely continuous on $[0,1]$ for $0\leqq q\leqq m$ has a sequence of Fourier coefficients g_k satisfying $g_k=o(1/n^{m+1})$. Conversely, if the Fourier coefficients of $g(\cdot)$ are bounded by $O(1/n^{m+2})$, then the derivatives $g^{(q)}(\cdot)$ exist and are continuous on $[0,1]$ for all q with $0\leqq q\leqq m$. The family of Poisson kernels

$$g^{\gamma}(x)=\frac{1-e^{-2\gamma}}{1-2\,e^{-\gamma}\cos 2\pi x+e^{-2\gamma}}$$

with Fourier coefficients $g_k^{\gamma}=e^{-\gamma|k|}$ emphasizes the relationship between the decay of the Fourier coefficients and the "global smoothness" of the curve since here, the larger γ is, the flatter g^{γ} gets.

It is then natural to look at penalties of the form

$$J_\alpha(g)=\sum_{k=-\infty}^{\infty}\alpha(k)^2|g_k|^2 \tag{6}$$

where α is a map from $\mathbb{R}$ to $\mathbb{R}$, which we will refer to as the weight function. Since, for a real-valued function $g(\cdot)$, we have $g_{-k}=g_k^*$ (where z^* is the complex conjugate of a complex number z), we assume

$$\alpha(-k)^2=\alpha(k)^2. \tag{A1}$$

Then let $\mathscr{C}_\alpha$ be the vector space of real-valued periodic functions $g(\cdot)$ with period 1, square integrable on $[0,1]$ and such that

$$J_\alpha(g)<\infty. \tag{E}$$

It is then easy to see that (4) is a particular case of (6). For $g(\cdot)$ in $W_m^{\mathrm{per}}(0,1)$, we have the following relationship between the Fourier coefficients of $g(\cdot)$ and those of the qth derivative, for $0\leqq q\leqq m$

$$g_k^{(q)}=(2\pi ik)^q g_k. \tag{7}$$

Using the Plancherel formula and (7), we obtain

$$\int_0^1|g^{(m)}(t)|^2\,dt=\sum_{k=-\infty}^{\infty}|g_k^{(m)}|^2=\sum_{k=-\infty}^{+\infty}|2\pi k|^{2m}|g_k|^2.$$

We can do the same transformation for a linear differential operator with constant coefficients, and find that the weight $\alpha(\cdot)$ is the modulus of its characteristic polynomial.

2. Construction of α-splines.

2.1. The spaces $\mathscr{C}_\alpha$. We first describe in more detail the spaces $\mathscr{C}_\alpha$. In order to allow low frequencies to be nonpenalized, we assume

$$\alpha(k)=0\Leftrightarrow|k|<h \tag{A2}$$

for a fixed integer h. Note that $h=1$ for (4). Moreover, we assume that

$$\text{(A3)} \qquad \sum_{k=h}^{\infty} \frac{1}{\alpha(k)^2} < \infty$$

to obtain continuity of evaluation functionals (2) on $\mathscr{C}_\alpha$. Lemma 2.1 shows that the local smoothness of elements of $\mathscr{C}_\alpha$ is related to the growth rate of the sequence $(\alpha(k))_{k\in N}$.

LEMMA 2.1. *If $\sum_{k=h}^{\infty} k^{2p}/\alpha(k)^2 < \infty$ for some nonnegative integer p, then any element of $\mathscr{C}_\alpha$ is equivalent to a continuous function with p continuous derivatives.*

Proof. The sequences $|k|^P/\alpha(k)$ and $\alpha(k)|g_k|$ being square summable, we have that the sequence $|k|^P|g_k|$ is absolutely summable, therefore $\sum_{k=-\infty}^{\infty} k^P g_k e^{2\pi ikx}$ is an absolutely convergent series that defines a continuous function. Consequently, $h(x) = \sum_{k=-\infty}^{\infty} g_k e^{2\pi ikx}$ is a continuous function with p continuous derivatives and has Fourier coefficients g_k. We conclude that $h(x)=g(x)$ almost everywhere, i.e., h and g are L_2-equivalent. □

For $p=0$, using (A3), we conclude that for any element $f(\cdot)$ in $\mathscr{C}_\alpha$, the sequence (f_k) is absolutely summable and the continuous function $\sum_{k=-\infty}^{\infty} f_k e^{2\pi ikx}$ is equal almost everywhere to $f(x)$. From now on, we will drop the distinction between the two.

We also conclude from Lemma 2.1 that, if $\alpha(\cdot)$ increases exponentially, then the elements of $\mathscr{C}_\alpha$ are infinitely differentiable functions.

It is clear that, if $\alpha(\cdot)>\beta(\cdot)$ from some point on, then $\mathscr{C}_\alpha \subset \mathscr{C}_\beta$. By letting $\alpha(k)=e^a(k^b/b)$, we obtain a variety of spaces between the usual Sobolev spaces and the finite-dimensional ("parametric") spaces of trigonometric polynomials of order less or equal than some integer p, which can be seen as a degenerate case where $\alpha(\cdot)$ is set to be ∞ from some point on.

Note that, for any $\alpha(\cdot)$, the set of restrictions to $[0, 1]$ of functions of $\mathscr{C}_\alpha$ is dense in L_2 norm, since $\mathscr{C}_\alpha$ always contains the space of trigonometric polynomials of all orders.

A reproducing kernel Hilbert space H is a Hilbert space in which evaluation functionals (2) are continuous for any t_i. Then, by Riesz representation theorem, for each t, there exists an element $l_t \in H$, called representer of evaluation at t, such that

$$\forall h \in H \quad L_t h = (l_t, h)_H$$

where $(\cdot,\cdot)_H$ denotes the inner product in H.

LEMMA 2.2. *With assumptions* (A1), (A2), *and* (A3), *$\mathscr{C}_\alpha$ is a reproducing kernel Hilbert space equipped with the inner product*

$$\text{(8)} \qquad \langle u(\cdot), v(\cdot)\rangle_{\mathscr{C}_\alpha} = \sum_{|k|<h} u_k v_k^* + \sum_{|k|\geqq h} \alpha(k)^2 u_k v_k^*$$

and the representer of evaluation at t is given by

$$\text{(9)} \qquad l_t(s) = l_0(s-t)$$

where l_0 is given by its Fourier coefficients

$$\text{(10)} \qquad (l_0)_k = \begin{cases} 1 & \textit{for } |k|<h, \\ 1/\alpha(k)^2 & \textit{for } |k|\geqq h. \end{cases}$$

Proof. It is clear that (8) defines an inner product in $\mathscr{C}_\alpha$, which can be decomposed into the orthogonal sum of the following two subspaces:

— $\mathscr{C}_\alpha^0$ is the space of trigonometric polynomials of order less than or equal to $h-1$, generated by 1, $\cos 2\pi kx$, $\sin 2\pi kx$, for $1 \leqq k \leqq h-1$.

— $\mathscr{C}_\alpha^1$ is the space of elements $f(\cdot)$ in $\mathscr{C}_\alpha$ such that $f_k = 0$ for $|k| < h$.

$\mathscr{C}_\alpha^1$ is isomorphic to l_2, the space of square summable sequences, by the map

$$\begin{aligned} &\mathscr{C}_\alpha \overset{T}{\rightarrow} l_2, \\ &f(\cdot) \rightarrow (f_k)_{k \geqq h}, \end{aligned} \tag{11}$$

and hence is complete, and so is $\mathscr{C}_\alpha^0$ as a finite-dimensional subspace. By (A3), the elements l_t defined by (9) and (10) belong to $\mathscr{C}_\alpha$, and we have

$$(l_t)_k = e^{-2\pi ikt}(l_0)_k$$

therefore, for any $f(\cdot)$ in $\mathscr{C}_\alpha$, using (8) we have that

$$\langle f(\cdot), l_t(\cdot)\rangle_{\mathscr{C}_\alpha} = \sum_{k=-\infty}^{\infty} f_k e^{2\pi ikt} = f(t). \qquad \square$$

2.2. Definition of the estimates. We now define the periodic smoothing α-spline relative to the data $y_1, \cdots, y_n$, to be the minimizer of

$$\mathrm{LS}(g) + J_\alpha(g) \tag{12}$$

over $g(\cdot) \in \mathscr{C}_\alpha$. The existence and uniqueness of the minimizer is guaranteed if $n \geqq 2h-1$, by a theorem of Anselone and Laurent [1] provided we check the following points:

(i) The range of the map T, defined by (11), is closed, which is clearly satisfied.

(ii) If $g(t_i) = 0$ for $i = 1, \cdots, n$ and if $Tg = 0$, then $g(\cdot)$ is a trigonometric polynomial of order less than or equal to $h-1$, vanishing at n points with $n \geqq 2h-1$, hence $g(\cdot) = 0$.

(iii) For any set of distinct points $t_1, \cdots, t_n$, the reproducing kernels $l_{t_1}, \cdots, l_{t_n}$ are linearly independent. To see this, it is enough to check that the matrix Σ with elements $l_{t_i}(t_j)$ is nonsingular. Using (9), we see that this follows from the fact that $l_0(\cdot)$ is a function of positive type, i.e., for any complex numbers $Z_i, \cdots, Z_n$,

$$\sum_{i=1}^{n}\sum_{i=1}^{n} Z_i l_0(t_i - t_j) Z_j^* = \sum_{k=-\infty}^{\infty} (l_0)_k \left|\sum_{i=1}^{n} Z_i e^{2\pi ikt_i}\right|^2$$

is greater than or equal to zero by positivity of the Fourier coefficients of l_0 given by (10). We then obtain a one-parameter (λ) family of estimates by fixing a weight function α_0 and considering either

$$\alpha_\lambda(k) = \lambda\alpha_0(k) \tag{13}$$

or

$$\alpha_\lambda(k) = \alpha_0(\lambda k). \tag{14}$$

These two ways coincide in the homogeneous case.

Theoretical formulas for the determination of these minimizers are formally similar to those for ordinary splines (see Kimeldorf and Wahba [5]).

3. Asymptotic bias and variance. The results of this section will be derived in the case when the design points t_i are equispaced:

$$t_i = \frac{i-1}{n}, \qquad i = 1, \cdots, n. \tag{15}$$

We make explicit the solution of the minimization problem and find closed formulas for the Fourier coefficients of the estimates, which we will use for the theoretical study of asymptotic bias and variance, as well as for the computations of § 6. We apply the classical techniques for this kind of problem, nicely reviewed for the case of ordinary periodic splines in Eubank [3]. Let $g^{n\lambda}$ denote the estimates obtained using the one-parameter family defined by (13) with observations at t_i defined by (15).

3.1. Computation of the estimates. For this computation, we momentarily drop the upper indices in $g^{n\lambda}$, and simply call it g.

The following notation will be used:

— For a sequence p_k and a given integer n, let $p_k^{(n)}$ be the periodic sequence defined by

$$p_k^{(n)} = \sum_{s=-\infty}^{\infty} p_{k+sn}. \tag{16}$$

— Let W (the finite Fourier transform) denote the matrix with elements

$$W_{rs} = n^{-1/2}\, e^{2\pi i(rs/n)}, \qquad r=0,\cdots,n-1, \quad s=0,\cdots,n-1.$$

It is easy to check that W is a unitary matrix, i.e., $W^t W^* = I$.

— Let G_n denote the vector with components $g(t_{in})$, for $i=1,\cdots,n$.

— Let Y_n denote the vector of data y_i, for $i=1,\cdots,n$.

The role of W is explained by the following important equation:

$$\begin{aligned} g(t_l) &= \sum_{\nu=-\infty}^{\infty} g_\nu\, e^{2\pi i\nu t_l} \\ &= \sum_{k=0}^{n-1}\sum_{s=-\infty}^{\infty} g_{k+sn}\exp\left(2\pi i(k+sn)\frac{l-1}{n}\right) \qquad (17)\\ &= \sum_{k=0}^{n-1} g_k^{(n)}\exp\left(2\pi i\frac{k(l-1)}{n}\right) = \sqrt{n}\sum_{k=0}^{n-1} g_k^{(n)} W_{k(l-1)}. \end{aligned}$$

The complex numbers $g_k^{(n)}$ are called aliased Fourier coefficients of $g(\cdot)$. Note that $g(\cdot)$ being real-valued, we have $g_{-k} = g_k^*$ and hence $g_{n-k}^{(n)} = g_k^{(n)*}$. So if we denote by $\tilde{G}_n$ the vector of aliased Fourier coefficients $g_k^{(n)}$ for $k=0,\cdots,n-1$, from (16) we have

$$\frac{1}{\sqrt{n}} G_n = W\tilde{G}_n$$

or equivalently

$$\tilde{G}_n = \frac{1}{\sqrt{n}}\, {}^tW^* G_n.$$

This motivates the introduction of the random vector $\tilde{Y}_n$ (with complex components) defined by

$$\tilde{Y}_n = \frac{1}{\sqrt{n}}\, {}^tW^* Y_n.$$

$\tilde{Y}_n$ will represent the vector of noisy measurements of the aliased Fourier coefficients.

Using the fact that W is unitary, we have

$$\left\| \frac{1}{\sqrt{n}} G_n - \frac{1}{\sqrt{n}} Y_n \right\|^2 = \|\tilde{G}_n - \tilde{Y}_n\|^2,$$

i.e.,

$$\frac{1}{n} \sum_{l=1}^{n} (g(t_l) - y_l)^2 = \sum_{l=1}^{n} |g_l^{(n)} - \tilde{y}_l|^2. \tag{18}$$

This formula is a discrete version of Plancherel.

We can now reformulate the minimization of (12) into the minimization of

$$F(g) = \sum_{l=1}^{n} |g_l^{(n)} - \tilde{y}_l|^2 + \sum_{|l| \geqq h} \alpha_l^2 |g_l| \tag{19}$$

over $g(\cdot)$ in $\mathscr{C}_\alpha$. Let us introduce the notation a_l:

$$\begin{aligned} a_l &= \alpha_l^2 \quad \text{for } |l| \geqq h, \\ &= 0 \quad \text{for } |l| < h. \end{aligned}$$

Using the calculus of variations, a solution $g(\cdot)$ will satisfy

$$\lim_{\varepsilon \to 0} \frac{1}{\varepsilon} [F(g + \varepsilon\phi) - F(g)] = 0$$

for any $\phi(\cdot) \in \mathscr{C}_\alpha$. Since

$$\begin{aligned} \frac{1}{\varepsilon}[F(g+\varepsilon\phi) - F(g)] = \frac{1}{\varepsilon} \sum_{l=1}^{n} & |g_l^{(n)} + \varepsilon\phi_l^{(n)} - \tilde{y}_l|^2 - |g_l^{(n)} - \tilde{y}_l|^2 \\ & + \sum_{l=-\infty}^{\infty} a_l[|g_l + \varepsilon\phi_l|^2 - |g_l|^2], \end{aligned}$$

we must have for all $\phi(\cdot) \in \mathscr{C}_\alpha$:

$$\sum_{l=1}^{n} \phi_l^{(n)}(g_l^{(n)*} - \tilde{y}_l^*) + \phi_l^{(n)*}(g_l^{(n)} - \tilde{y}_l) + \sum_{l=-\infty}^{\infty} a_l(\phi_l^* g_l + \phi_l g_l^*) = 0$$

hence

$$\sum_{k=0}^{n-1} \sum_{s=-\infty}^{\infty} \phi_{k+sn}(g_k^{(n)*} - \tilde{y}_k^*) + \phi_{k+ns}^*(g_k^{(n)} - \tilde{y}_k) + a_{k+ns}(\phi_{k+ns}^* g_{k+ns} + \phi_{k+ns} g_{k+ns}^*) = 0.$$

Since $\phi(\cdot)$ is arbitrary in $\mathscr{C}_\alpha$ (in particular, ϕ can be any trigonometric polynomial), we conclude that

$$g_k^{(n)} - \tilde{y}_k + a_{k+sn} g_{k+sn} = 0 \quad \forall k = 0, \cdots, n-1 \quad \forall s. \tag{20}$$

— If we first take k such that $|k| < h$, then $\alpha_k = 0$, hence

$$g_k^{(n)} = \tilde{y}_k \quad \text{and} \quad g_{k+sn} = 0 \quad \forall s \neq 0$$

so

$$g_k = g_k^{(n)} = \tilde{y}_k \quad \forall |k| < h. \tag{21}$$

— If now $|n-k| < h$, then $a_{k-n} = 0$, hence

$$g_k^{(n)} = \tilde{y}_k \quad \text{and} \quad g_{k+sn} = 0 \quad \forall s \neq -1$$

so

$$g_{k-n} = g_k^{(n)} = \tilde{y}_k \quad \forall |n-k| < h. \tag{22}$$

Note that, if $n = 2h-1$, for all $k = 1, \cdots, n$, we have either $|k| < h$ or $|n-k| < h$ and hence in that case, $g(\cdot)$ is a trigonometric polynomial.

— For all other k, i.e., $h \leqq k \leqq n-h$, there is no s such that $k+ns \in (-h, h)$ so that $a_{k+sn} \neq 0$ for all s and

$$g_{k+ns} = a_{k+ns}^{-1}(\tilde{y}_k - g_k^{(n)}). \tag{23}$$

Sum on s to get

$$g_k^{(n)} = \sum_{s=-\infty}^{\infty} a_{k+ns}^{-1}(\tilde{y}_k - g_k^{(n)}),$$

and hence

$$g_k^{(n)} = \frac{\sum_{s=-\infty}^{\infty} a_{k+ns}^{-1}}{1+\sum_{s=-\infty}^{\infty} a_{k+ns}^{-1}} \tilde{y}_k.$$

Replacing in (20) gives

$$g_{k+sn} = \frac{a_{k+ns}^{-1}}{1+\sum_{q=-\infty}^{\infty} a_{k+qn}^{-1}} \tilde{y}_k, \qquad h \leqq k \leqq n-h. \tag{24}$$

Equations (21)-(23) describe entirely the unique solution in terms of its Fourier coefficients.

Denoting by $b_l = a_l^{-1}$ for $|l| \geqq h$, we can write

$$g_{k+sn} = \frac{b_{k+sn}}{1+b_k^{(n)}} \tilde{y}_k \tag{25}$$

and with the conventions that $b_l = \infty$ for $|l| < h$ and $\infty/\infty = 1$ and $b_{l+sn} = 0$ for $s \neq 0$ and $|l| < h$, the formula is true for all indices.

3.2. Eigenvalue analysis. Let I_Y denote the interpolating ($\lambda = 0$) α-spline relative to the data Y. $J(I_Y)$ is a quadratic form in the data vector Y, whose matrix we denote by Ω. Easy algebraic arguments show that the estimate $g^{n\lambda}(\cdot)$ is then the interpolating spline relative to the fitted vector $\tilde{Y} = (I + n\lambda\Omega)^{-1} Y$. If we denote by μ_k, $k = 1, \cdots, n$, the eigenvalues of Ω and by $V_1, \cdots, V_n$ an orthonormal basis of eigenvectors of Ω, then we can rewrite the estimate as

$$g^{n\lambda}(\cdot) = \sum_{k=1}^{n} \frac{1}{1+n\lambda\mu_k} \gamma_k I_{V_k}(\cdot) \quad \text{for } Y = \sum_{k=1}^{n} \gamma_k V_k. \tag{26}$$

This formula emphasizes the "tapered Fourier series" aspect of the estimate. In the equispaced design points case, explicit formulas for μ_k and I_{V_k} are available, and they relate the damping factor $1/(1+n\lambda\mu_k)$ to the weight function α:

$$\begin{aligned} &\mu_k = \frac{1}{nb_k^{(n)}}, && h \leqq k \leqq n, \\ &\mu_k = 0, && 1 \leqq k \leqq h, \\ &I_{V_k}(t) = n^{-1/2} e^{2\pi ikt}\left(\sum_{s=-\infty}^{\infty} \frac{b_{k+sn}}{b_k^{(n)}} e^{2\pi iust}\right). \end{aligned} \tag{27}$$

3.3. Bias. The first issue in the assessment of these estimates is to determine whether they are asymptotically unbiased, i.e., whether the pointwise bias

$$b^{n\lambda}(t) = Eg^{n\lambda}(t) - f(t) \tag{28}$$

converges to zero when n converges to ∞, in some norm and for some sequence of smoothing parameters.

The following theorem proves that, when $\mathscr{C}_\alpha \subset W_2^{\text{per}}$, $g^{n\lambda}(\cdot)$ is asymptotically unbiased in the norm of $\mathscr{C}_\alpha$ if the true function $f(\cdot)$ belongs to $\mathscr{C}_\alpha$, for an appropriate sequence of smoothing parameters λ_n, and furthermore that if we overestimate a priori the actual smoothness of the true function by using a sequence of weights α_n increasing too fast for $f(\cdot)$, then the estimate is still asymptotically unbiased in W_2^{per}-norm provided $f(\cdot)\in W_2^{\text{per}}$ and $\mathscr{C}_\alpha \subset W_2^{\text{per}}$.

THEOREM 3.1. *Assuming that $\lambda\to 0$ and $n^2\lambda\to\infty$ as $n\to\infty$, and that $\alpha_k \geqq \gamma k^2$ for all k, for some positive γ, we have the following*:

(i) *If $f(\cdot)\in\mathscr{C}_\alpha$, then $\|Eg^{n\lambda}(\cdot)-f(\cdot)\|_{\mathscr{C}_\alpha}\to 0$ as $n\to\infty$.*

(ii) *If $f(\cdot)\in W_2^{\text{per}}$, then $\|Eg^{n\lambda}(\cdot)-f(\cdot)\|_{W_2^{\text{per}}}\to 0$ as $n\to\infty$.*

Proof. Let

(29) $$T_n(g):=\frac{1}{n}\sum_{i=1}^{n}(g(t_{in})-f(t_{in}))^2+\lambda\sum_{|p|\geqq h}\alpha_p^2|g_p|^2,$$

(30) $$T(g):=\int_0^1 (g(t)-f(t))^2\,dt+\lambda\sum_{|p|\geqq h}\alpha_p^2|g_p|^2.$$

$f^{n\lambda}=E(g^{n\lambda})$ is the minimizer of $T_n(g)$ over $g(\cdot)\in\mathscr{C}_\alpha$, by linearity of expectation and linearity of the solution of such a minimization problem. Moreover, the calculus of variations shows that, for $f(\cdot)\in L_2(0,1)$, there exists a unique element $f^\lambda(\cdot)$ minimizing $T(g)$ over $g(\cdot)\in\mathscr{C}_\alpha$, and it is given by its Fourier coefficients:

(31) $$\begin{aligned} f_k^\lambda &= f_k && \text{for } |k|<h,\\ &=\frac{1}{1+\lambda\alpha_k^2}f_k && \text{for } |k|\geqq h.\end{aligned}$$

A first step in comparing $f(\cdot)$ to $f^{n\lambda}(\cdot)$ is to compare $f(\cdot)$ to f^λ.

LEMMA 3.1. (i) *If $f(\cdot)\in W_2^{\text{per}}$ then $\|f^\lambda(\cdot)-f(\cdot)\|_{W_2^{\text{per}}}\to 0$ as $\lambda\to 0$.*

(ii) *If $f(\cdot)\in\mathscr{C}_\alpha$ then $\|f^\lambda(\cdot)-f(\cdot)\|_{\mathscr{C}_\alpha}\to 0$ as $\lambda\to 0$.*

Proof. We will take

$$\|f(\cdot)\|_{W_2^{\text{per}}}=\sum_{k=-\infty}^{\infty}(1+|k|^4)|f_k|^2$$

as a norm in W_2^{per}. Using (31), we see that

$$\|f^\lambda(\cdot)-f(\cdot)\|^2_{W_2^{\text{per}}}=\sum_{\substack{k=-\infty\\|k|\geqq m}}^{\infty}\left(\frac{\lambda\alpha_k^2}{1+\lambda\alpha_k^2}\right)^2(1+|k|^4)|f_k|^2.$$

Then (i) results from the dominated convergence theorem and the argument is clearly similar for (ii). $\square$

Now it remains to compare $f^{n\lambda}(\cdot)$ to $f^\lambda(\cdot)$.

LEMMA 3.2. *For any $g(\cdot)\in\mathscr{C}_\alpha$ and $\lambda\leqq 1$,*

$$\|g(\cdot)-f^\lambda(\cdot)\|^2_{\mathscr{C}_\alpha}\leqq\frac{1}{\lambda}(T(g)-T(f^\lambda)).$$

Proof.

$$\begin{aligned}0&\leqq T(g)-T(f^\lambda)\\ &=\sum_{k=-\infty}^{\infty}\{|g_k-f_k|^2-|f_k^\lambda-f_k|^2+\lambda a_k(|g_k|^2-|f_k^\lambda|^2)\}\\ &=\sum_{k=-\infty}^{\infty}\{g_kg_k^*-f_k^*(g_k-f_k^\lambda)-f_k(g_k^*-f_k^{\lambda*})\\ &\qquad\qquad -f_k^\lambda f_k^{\lambda*}+\lambda a_k(g_kg_k^*-f_k^\lambda f_k^{\lambda*})\}\end{aligned}$$

and using (31)

$$= \sum_{k=-\infty}^{\infty} \{g_k g_k^* - (1+\lambda a_k) f_k^\lambda (g_k^* - f_k^{\lambda *}) - (1+\lambda a_k) f_k^{\lambda *}(g_k - f_k^\lambda) - f_k^\lambda f_k^{\lambda *} + \lambda a_k (g_k g_k^* - f_k^\lambda f_k^{\lambda *})\}$$

$$= \sum_{l=-\infty}^{\infty} (1+\lambda a_k)\{g_k g_k^* - f_k^\lambda (g_k^* - f_k^{\lambda *}) - f_k^{\lambda *}(g_k - f_k^\lambda) - f_k^f f_k^{\lambda *}\}$$

$$= \sum_{k=-\infty}^{\infty} (1+\lambda a_k)|g_k - f_k^\lambda|^2 \geqq \lambda \|g(\cdot) - f^\lambda(\cdot)\|_{\mathscr{C}_\alpha}^2. \qquad \square$$

Before proving Theorem 3.1, we also need to compare $T_n(g)$ to $T(g)$ for $g(\cdot) \in \mathscr{C}_\alpha$. Note that for (i) and (ii), $f(\cdot)$ belongs to W_2^{per}. We will use the classical quadrature approximation:

$$\forall h(\cdot) \in W_2^{\text{per}}, \quad \left| \frac{1}{n} \sum_{i=1}^{n} h(t_{in})^2 - \int_0^1 h(t)^2\, dt \right| \leqq \frac{c}{n^2} \|h(\cdot)\|_{W_2^{\text{per}}}^2 \tag{32}$$

from which we conclude that there exists a constant C such that, for any $g(\cdot)$ in $\mathscr{C}_\alpha$,

$$|T_n(g) - T(g)| \leqq \frac{C}{n^2} \|g(\cdot) - f(\cdot)\|_{W_2^{\text{per}}}^2. \tag{33}$$

Lemma 3.1 compares $f(\cdot)$ to $f^\lambda(\cdot)$ in the appropriate norms, so it remains to compare $f^\lambda(\cdot)$ to $f^{n\lambda}(\cdot)$. Using the optimality of f^λ, we have

$$T_n(f^{n\lambda}) - T(f^{n\lambda}) \leqq T_n(f^{n\lambda}) - T(f^\lambda). \tag{34}$$

Using the optimality of $f^{n\lambda}$, we have

$$T_n(f^{n\lambda}) - T(f^\lambda) \leqq T_n(f^\lambda) - T(f^\lambda). \tag{35}$$

Hence, combining this with (33):

$$\begin{aligned} &|T_n(f^{n\lambda}) - T(f^\lambda)| \leqq \sup(|T_n(f^{n\lambda}) - T(f^{n\lambda})|, |T_n(f^\lambda) - T(f^\lambda)|), \\ &|T_n(f^{n\lambda}) - T(f^\lambda)| \leqq \frac{C}{n^2} \sup(\|f^{n\lambda} - f\|_{W_2^{\text{per}}}^2, \|f^\lambda - f\|_{W_2^{\text{per}}}^2). \end{aligned} \tag{36}$$

Since the $\mathscr{C}_\alpha$-norm dominates the W_2^{per}-norm, we get

$$\begin{aligned} \|f^{n\lambda} - f\|_{W_2^{\text{per}}}^2 &\leqq 2[\|f^{n\lambda} - f^\lambda\|_{\mathscr{C}_\alpha}^2 + \|f^\lambda - f\|_{W_2^{\text{per}}}^2] \\ &\leqq \frac{2}{\lambda}[|T(f^{n\lambda}) - T_n(f^{n\lambda})| + |T_n(f^{n\lambda}) - T(f^\lambda)|] + 2\|f^\lambda - f\|_{W_2^{\text{per}}}^2. \end{aligned}$$

Using (36) and (33) again, we have

$$\|f^{n\lambda} - f\|_{W_2^{\text{per}}}^2 \leqq \frac{4C}{\lambda n^2}[\sup(\|f^{n\lambda} - f\|_{W_2^{\text{per}}}^2, \|f^\lambda - f\|_{W_2^{\text{per}}}^2)] + 2\|f^\lambda - f\|_{W_2^{\text{per}}}^2.$$

Therefore, for n large enough,

$$\|f^{n\lambda} - f\|_{W_2^{\text{per}}}^2 \leqq \frac{2}{1 - 4C/\lambda n^2} \|f^\lambda - f\|_{W_2^{\text{per}}}^2.$$

Thus (i) results from this inequality and Lemma 3.1. For (ii), the argument is similar but requires the application of (i). □

A corollary to this proof is, by (33) and (i), that

$$T_n(f^{n\lambda}) - T(f^{n\lambda}) = O\left(\frac{1}{n^2}\right).$$

Therefore the integrated square bias

$$b_n^2(\lambda) = \int_0^1 (Eg^{n\lambda}(t) - f(t))^2\, dt \tag{37}$$

can be approximated by a discrete version

$$c_n^2(\lambda) = \frac{1}{n}\sum_{i=1}^{n} (f^{n\lambda}(t_i) - f(t_i))^2 \tag{38}$$

and the error of approximation is

$$c_n^2(\lambda) - b_n^2(\lambda) = T_n(f^{n\lambda}) - T(f^{n\lambda}) = O\left(\frac{1}{n^2}\right). \tag{39}$$

Moreover, using (18) and $E\tilde{y}_k = f_k^{(n)}$, we get

$$c_n^2(\lambda) = \sum_{k=0}^{n-1} |Eg_k^{(n)} - f_k^{(n)}|^2 = \sum_{k=0}^{n-1} \left(\frac{\lambda}{\lambda + b_k^{(n)}}\right)^2 |f_k^{(n)}|^2. \tag{40}$$

3.4. Mean square error. The bias is clearly an increasing function of λ, whereas the variance is a decreasing function of λ. Hence the choice of λ, which reflects the balance between fit to the data and smoothness, will determine the trade-off between bias and variance if we wish to minimize the mean square error:

$$R_n(\lambda) = \int_0^1 E(g^{n\lambda}(t) - f(t))^2\, dt = b_n^2(\lambda) + \int_0^1 \operatorname{var} g^{n\lambda}(t)\, dt. \tag{41}$$

The integrated variance $v_n(\lambda)$ can be approximated by a discrete version

$$\mu_n(\lambda) = \frac{1}{n}\sum_{i=1}^{n} \operatorname{var} g^{n\lambda}(t_i) = \frac{\sigma^2}{n}\sum_{k=0}^{n-1}\left(\frac{b_k^{(n)}}{\lambda + b_k^{(n)}}\right)^2 \tag{42}$$

and the following lemma evaluates the approximation error.

LEMMA 3.3. *If $\alpha(k) \geqq \gamma k^2$ for some γ asymptotically, then*

$$v_n(\lambda) - \mu_n(\lambda) = O\left(\frac{1}{n^2\lambda}\right) + O\left(\frac{1}{n^3}\right). \tag{43}$$

Proof. $v_n(\lambda)$ can be easily computed using (26)

$$v_n(\lambda) = \sigma^2 \sum_{k=0}^{n-1}\left(\frac{b_k^{(n)}}{\lambda + b_k^{(n)}}\right)^2 \int_0^1 \|I_{V_k}(t)|^2\, dt.$$

Therefore, using (32), we have

$$\begin{aligned} |v_n(\lambda) - \mu_n(\lambda)| &\leqq \frac{\sigma^2 C}{n^2}\sum_{k=0}^{n-1}\left(\frac{b_k^{(n)}}{\lambda + b_k^{(n)}}\right)^2 \|I_{V_k}(\cdot)\|^2 \mathscr{C}_\alpha \\ &\leqq \frac{\sigma^2 C}{n^2}\left\{\sum_{k=0}^{h}\frac{1}{n} + \sum_{k=h+1}^{n-1}\left(\frac{b_k^{(n)}}{\lambda + b_k^{(n)}}\right)\frac{1}{nb_k^{(n)}}\right\} \\ &\leqq \frac{\sigma^2 C}{n^2}\left(\frac{h}{n} + \frac{1}{\lambda}\right). \end{aligned}$$

□

Speckman [10], [11] established that D^m-splines reach an (optimal) rate of convergence, i.e., if $a_k = k^m$, m being a positive integer, and $\lambda_n = O(n^{2m/(2m+1)})$, then $\mathrm{IMSE}_n(\lambda) = O(n^{2m/(2m+1)})$. We prove here that for an exponentially growing sequence of weights α_n we obtain faster rates of convergence when the true function belongs to the corresponding space $\mathscr{E}_\alpha$.

THEOREM 3.2. *If the weights α_k are asymptotically equivalent to $\gamma e^{\xi k^\nu}$ for $k \to \infty$, for some positive reals ξ and ν, then there exists a sequence of smoothing parameters λ_n such that* $\mathrm{IMSE}_n(\lambda_n) = O((\log n)^{1/\nu}/n)$.

Proof. Using (39), we first need to evaluate the size of $c_n^2(\lambda)$: when $f(\cdot)$ belongs to $\mathscr{E}_\alpha$, using the optimality of $f^{n\lambda}$, we can write

$$c_n^2(\lambda) \leqq T_n(f^{n\lambda}) \leqq T_n(f) = \lambda \sum_{|k| \geqq h} \alpha_k^2 |f_k|^2$$

hence if $f(\cdot) \in \mathscr{E}_\alpha$, then $c_n^2(\lambda) = O(\lambda)$, we conclude that

$$b_n^2(\lambda) = O(\lambda) + O\left(\frac{1}{n^2}\right) + O\left(\frac{1}{n^2\lambda}\right). \tag{44}$$

Turning to variance, by symmetry ($b_k^{(n)} = b_{n-k}^{(n)}$), we can consider only those k between 0 and $[n/2]$. Since $b_k^{(n)}/(\lambda + b_k^{(n)})$ is always bounded by 1, any finite fixed number of terms will contribute for $O(1/n)$, so if for $k > K$ we have $A\, e^{\xi k^\nu} \leqq \alpha_k \leqq B\, e^{\xi k^\nu}$, we can concentrate on the terms for which $K < k \leqq [n/2]$.

We claim that there exists a constant M and an integer N such that for all $n \geqq N$ and $K \leqq k \leqq [n/2]$ we have

$$b_k \leqq b_k^{(n)} \leqq M b_k. \tag{45}$$

It is clear that $b_k^{(n)} \geqq b_k$. For $k \geqq K$, we have $(1/B^2)\, e^{-2\xi k^\nu} \leqq b_k \leqq (1/A^2)\, e^{-2\xi k^\nu}$. Hence

$$0 \leqq \frac{b_k^{(n)} - b_k}{b_k} \leqq \frac{B^2}{A^2} e^{2\xi k^\nu} \sum_{s \neq 0} e^{-2\xi |k+sn|^\nu}$$

$$\leqq \frac{B^2}{A^2} \left\{ \sum_{s=1}^{\infty} e^{-2\xi[(k+sn)^\nu - k^\nu]} + \sum_{s=1}^{\infty} e^{-2\xi[(sn-k)^\nu - k^\nu]} \right\}$$

since

$$(sn+k)^\nu - k^\nu \geqq n^\nu(s^\nu - (\tfrac{1}{2})^\nu) \text{ and } (sn-k)^\nu - k^\nu \geqq n^\nu((s-\tfrac{1}{2})^\nu - (\tfrac{1}{2})^\nu) \text{ for } 0 \leqq k/n \leqq \tfrac{1}{2},$$

$$(b_k^{(n)} - b_k)/b_k \leqq B^2/A^2 \sum_{s=1}^{\infty} e^{-2\xi n^\nu (s^\nu - (1/2)^\nu)} + e^{-2\xi n^\nu((s-1/2)^\nu - (1/2)^\nu)}$$

choose a β such that $\nu\beta > 1$, and choose n large enough so that

$$\frac{b_k^{(n)} - b_k}{b_k} \leqq \frac{B^2}{A^2 (2\xi)^\beta n^{\nu\beta}} \left\{ \sum_{s=1}^{\infty} \frac{1}{(s^\nu - (\frac{1}{2})^\nu)^\beta} + \sum_{s=2}^{\infty} \frac{1}{(s-(\frac{1}{2}))^\nu - ((\frac{1}{2})^\nu)^\beta} + 1 \right\},$$

which proves that this quantity is bounded by a constant M independent of k and n.

Returning to the theorem, we have

$$\frac{\sigma^2}{n} \sum_{K \leqq k \leqq [n/2]} \left(\frac{b_k^{(n)}}{\lambda + b_k^{(n)}} \right)^2 \leqq \frac{\sigma^2}{n} \sum_{K \leqq k \leqq [n/2]} \frac{1}{(1 + \lambda A^2\, e^{2\xi k^\nu}/M)^2}.$$

Now divide this sum into two parts according to k:

— In the first part, we take all k's such that $\xi k^\nu < \log(1/\lambda)$, and we bound each summand by one so that the first sum is bounded by

$$\frac{\sigma^2}{n} \left(\frac{1}{\xi} \log\left(\frac{1}{\lambda}\right) \right)^{1/\nu}.$$

—In the second part, each k satisfies $\lambda e^{2\xi k^\nu} > e^{\xi k^\nu}$, hence the second sum is bounded by

$$\frac{\sigma^2}{n} \sum_{k=1}^{\infty} \frac{1}{(1+A^2 e^{\xi k^\nu}/M)^2},$$

which is $O(1/n)$. If we now take the sequence of smoothing parameters $\lambda_n = (\log n)^{1/\nu}/n$, we get that the first part is $O((\log n)^{1/\nu}/n)$. Using (44) this concludes the proof since all other contributions to the IMSE are either faster, such as $O(1/n)$, $O(1/n^2)O(1/n^2\lambda)$, $O(1/n^3)$, or similar like $O(\lambda)$. □

Note that this statement only gives a bound for the rate and there is no optimality property.

4. Equivalence with kernel estimation. As a linear estimate, we can rewrite $g^{n\lambda}(\cdot)$

$$g^{n\lambda}(t) = \frac{1}{n} \sum_{i=1}^{n} y_i G(t, t_i) \tag{46}$$

and compare the impulse response functions $G(\cdot, t_i)$ to those corresponding to kernel estimates

$$G(t, t_i) = \frac{1}{b} K\left(\frac{t-t_i}{b}\right)$$

for a kernel K and a bandwidth b.

Several connections between splines and kernels have been established by Cogburn and Davis [2] for periodic L-splines and Silverman [9] for D^m-splines.

Silverman shows that, by blowing up the t-axis near t_0, and properly renormalizing to obtain a nondegenerate limit, it is possible to approximate asymptotically $G(t, t_i)$ by a kernel-type impulse response with a known kernel κ_m, and bandwidth depending on the smoothing parameter λ and the limiting local density of design points. If, for simplicity, we specialize his result to the equispaced data points case, the precise statement is that $\lambda^{1/2m} G(t+\lambda^{1/2m} t_0, t)$ converges to $\kappa_m(t_0)$ when $n \to \infty$ and $\lambda \to 0$ and $n^{2m}\lambda \to \infty$.

Cogburn and Davis [2] give bounds for the norms $(\mathcal{L}_\infty, \mathcal{L}_1, \mathcal{L}_2)$ of the difference between $t \to G(t, t_i)$ and $t \to (1/\lambda^{1/2m})\kappa_m((t-t_i)/\lambda^{1/2m})$ in the periodic equispaced case.

We will generalize Cogburn and Davis's result to the case of periodic α-splines. Here we combine (13) and (14) in

$$\alpha(k) = \lambda \alpha(\mu k) \tag{47}$$

for $k \geqq h$ and $\alpha(k) = 0$ for $k < h$. The first goal is to give explicitly the impulse response functions $G(t, t_i)$. By shift invariance property of the estimator, they only depend on the difference $t - t_i$, so let

$$G(t, t_i) = G_{n\lambda\mu}(t - t_i).$$

Using (25), we can write

$$g^{n\lambda\mu}(t) = \sum_{k=0}^{n-1} \sum_{s=-\infty}^{\infty} \frac{b_{k+sn}(\mu)}{\lambda + b_k^{(n)}(\mu)} \tilde{y}_k e^{2\pi i(k+2n)t}$$

where $b_l(\mu) = 1/\alpha^2(\mu l)$ for $l = k+sn$, $|k| \geqq h$ and $b_l(\mu) = \infty$ for $|l| < h$, $b_l(\mu) = 0$ for $l = k+sn$, $s \neq 0$ and $|l| < h$. Replacing $\tilde{Y}$ in terms of the original data $Y = (y_1, \cdots, y_n)'$,

we obtain

$$g^{n\lambda\mu}(t)=\sum_{k=0}^{n-1}\sum_{s=-\infty}^{\infty}\sum_{l=0}^{n-1}\frac{1}{n}y_l\,e^{-2\pi i(kl/n)}\,e^{2\pi i(k+sn)t}\frac{b_{k+sn}(\mu)}{\lambda+b_k^{(n)}(\mu)}$$
$$=\frac{1}{n}\sum_{l=0}^{n-1}y_l\left\{\sum_{k=0}^{n-1}\sum_{s=-\infty}^{\infty}\frac{b_{k+sn}(\mu)}{\lambda+b_k^{(n)}(\mu)}e^{2\pi i(k+sn)(t-l/n)}\right\}.$$

This establishes the fact, with $G_{n\lambda\mu}$ being defined by its Fourier coefficients:

$$(G_{n\lambda\mu})_{k+sn}=\frac{b_{k+sn}}{\lambda+b_k^{(n)}}.$$

In an attempt to get an asymptotic expression for $G_{n\lambda\mu}$, it is natural to consider the continuous version of the minimization problem (see (31)), whose solution can be seen to be the convolution of $f(\cdot)$ by $G_{\lambda\mu}$ given by

$$(G_{\lambda\mu})_l=\frac{1}{1+\lambda\alpha^2(\mu l)}$$

for $|l|>h$, and $(G_{\lambda\mu})_l=1$ for $|l|<h$.

THEOREM 4.1. *Assuming the function $\alpha(\cdot)$ satisfies $\alpha(x)\geqq\gamma x^\nu$ for $|x|>A$ and $\gamma>0$, $\nu>\frac{1}{2}$, we have*

$$\|G_{n\lambda\mu}(\cdot)-G_{\lambda\mu}(\cdot)\|_\infty=O\left(\frac{1}{\lambda\mu^{2\nu}n^{2\nu-1}}\right)$$

and

$$\|G_{n\lambda\mu}(\cdot)-G_{\lambda\mu}(\cdot)\|^2_{\mathscr{L}_2(0,1)}=O\left(\frac{1}{\lambda^2\mu^{4\nu}n^{4\nu-1}}\right)$$

when $n\to\infty$, $\lambda\mu^{2\nu}n^{2\nu-1}\to\infty$, and $\mu n\to\infty$.

Proof. We can rewrite

$$(G_{\lambda\mu})_l=\frac{b_l(\mu)}{\lambda+b_l(\mu)}\quad\text{and}\quad(G_{n\lambda\mu})_l=\frac{b_l(\mu)}{\lambda+b_{\bar{l}}^{(n)}(\mu)}$$

where $\bar{l}=l \bmod n$ if $l>0$ and $=-l \bmod n$ if $l<0$. Then

$$\|G_{n\lambda\mu}(\cdot)-G_{\lambda\mu}(\cdot)\|_\infty\leqq\sum_{l=-\infty}^{\infty}\left|\frac{b_l(\mu)}{\lambda+b_l(\mu)}-\frac{b_l(\mu)}{\lambda+b_{\bar{l}}^{(n)}(\mu)}\right|$$
$$\leqq 2\sum_{h\leqq l\leqq[n/2]}\frac{b_l(\mu)|b_l^{(n)}(\mu)-b_l(\mu)|}{(\lambda+b_l(\mu))(\lambda+b_l^{(n)}(\mu))}+4\sum_{l>[n/2]}\frac{b_l(\mu)}{\lambda+b_l(\mu)}$$

we choose n large enough so that for $l>[n/2]$, $\alpha(\mu l)\geqq(\mu l)^\nu$, which is possible because $\mu n\to\infty$, then we have

—The second sum is bounded by $(1/\lambda)\sum_{l>[n/2]}(1/\alpha^2(\mu l))$, which is easily seen to be $O(1/\lambda\mu^{2\nu}n^{2\nu-1})$.

—The first sum is bounded by $(1/\lambda)\sum_{h\leqq l\leqq[n/2]}|b_l^{(n)}(\mu)-b_l(\mu)|$ but

$$|b_l^{(n)}(\mu)-b_l(\mu)|\leqq\frac{1}{\gamma^2\mu^{2\nu}}\sum_{s\neq 0}\frac{1}{(l+sn)^{2\nu}}\leqq\frac{1}{\gamma^2\mu^{2\nu}n^{2\nu}}\sum_{s=1}^{\infty}\frac{1}{(s-\frac{1}{2})^{2\nu}},$$

and hence the first sum is seen to be $O(1/\lambda\mu^{2\nu}n^{2\nu-1})$. The argument is similar for the $\mathscr{L}_2(0,1)$-norm. □

For a function $K(\cdot)$ on $\mathbb{R}$ define its "folded-back" version $K^f(\cdot)$ to be the periodic function:

$$K^f(x)=\sum_{s=-\infty}^{\infty} K(x-s).$$

Then there is a relationship between the Fourier transform of K and the Fourier coefficients of K^f given by

$$\hat{K}(n)=K_n^f \quad \text{for any integer } n$$

and if $\bar{h}$ denotes the periodic extension of a function h on $(0,1)$ to $\mathbb{R}$, then

$$\int_{-\infty}^{\infty} K(x-s)\bar{h}(s)\,ds=\int_0^1 K^f(x-s)h(s)\,ds$$

and similarly for discrete convolution (when $t_i=i/n$ for $i=0,\cdots,n$)

$$\frac{1}{n}\sum_{l=-\infty}^{\infty} K(x-t_l)\bar{h}(t_l)=\frac{1}{n}\sum_{i=1}^{n} K^f(x-t_i)h(t_i).$$

By examining the Fourier coefficients $(G_{\lambda\mu})_l$ of $G_{\lambda\mu}$, we conclude that they agree (at least for $|l|\geqq h$), with those of the "folded-back" version of $(1/\mu)K_\lambda(\cdot/\mu)$ where K_λ is the inverse Fourier transform of $1/(1+\lambda\alpha^2(\omega))$. If $h=1$, they agree for all l. If $h\neq 1$, and $\alpha(\cdot)$ vanish in an interval $(-\delta,\delta)$, then they agree for all l if μ is sufficiently small ($\mu<\delta/h$). If $h\neq 1$ and $\alpha(\cdot)$ vanish only at zero, the difference between $G_{\lambda\mu}(\cdot)$ and the folded-back version of $(1/\mu)K_\lambda(\cdot/\mu)$ is easily seen to converge to zero when $\mu\to 0$.

Hence the previous theorem shows that the impulse response function for α-splines are well approximated for large sample size by a kernel-type impulse function, when we use the "folded-back" version of the kernel (i.e., apply the chosen kernel to the periodic extension of the data points), which seems to be the reasonable thing to do in the periodic case. Moreover, we can prove that, for small bandwidth (i.e., μ here), there is very little difference between the original kernel and its "folded-back" version, at least for nice kernels, as the next lemma shows.

LEMMA 4.1. *If $\alpha(\cdot)$ is such that $1/(1+\lambda\alpha^2(\cdot))$ has p integrable and absolutely continuous derivatives ($p\geqq 2$), then*

$$\sup_{x\in[-\frac{1}{2},\frac{1}{2}]}\left|G_{\lambda\mu}(x)-\frac{1}{\mu}K_\lambda\left(\frac{x}{\mu}\right)\right|\to 0 \quad \text{as } \mu\to 0.$$

Proof. By the assumption on α, we know that $|x^pK_\lambda(x)|\leqq A_\lambda$ for some constant $A_\lambda(x^pK_\lambda(x)$ can be written as the Fourier transform of an integrable function $K_\lambda^{(p)}$ and therefore is bounded). Hence

$$\left|G_{\lambda\mu}(x)-\frac{1}{\lambda}K_\lambda\left(\frac{x}{\mu}\right)\right|\leqq\sum_{\substack{s=+\infty\\ s\neq 0}}^{\infty}\frac{1}{\mu}\left|K_\lambda\left(\frac{x-s}{\mu}\right)\right|$$

$$\leqq\sum_{\substack{s=-\infty\\ s\neq 0}}\frac{1}{\mu}\frac{A_\lambda}{|(x-s)/\mu|^p}$$

by symmetry it is enough to look at $x\in[0,\frac{1}{2}]$

$$\sup_{x\in[0,\frac{1}{2}]}\left|G_{\lambda\mu}(x)-\frac{1}{\mu}K_\lambda\left(\frac{x}{\mu}\right)\right|\leqq 2A_\lambda\mu^{p-1}\left(\sum_{s=1}^{\infty}\frac{1}{(s-\frac{1}{2})^p}\right),$$

which concludes the argument. □

We can check that K_λ agrees with κ_m for the choice $\alpha(\omega)=\omega^m$. Some other examples are:

— For $\lambda=1$ and $\alpha^2(\omega)=e^{2\pi^2\omega^2}-1$, then K_λ is a normal kernel $(1/\sqrt{2\pi})\, e^{x^2/2}$.

— For $\lambda=1$ and $\alpha^2(\omega)=\cosh(\omega/2)-1$, then $K_\lambda(x)=\pi/\cosh 2\pi^2 x$. Since K_λ and α are related by

$$\hat{K}_\lambda(\omega)=\frac{1}{1+\lambda\alpha^2(\omega)}, \tag{48}$$

we thus get all kernels $K(\cdot)$ whose Fourier transform $\hat{K}(\cdot)$ satisfy $0\leqq \hat{K}(\omega)\leqq 1$. They have the characteristics of "low-pass filters." The normalization $\hat{K}(0)=1(\alpha(0)=0)$ corresponds to $\int K(t)\,dt=1$. The number of finite moments of K corresponds to the number of times α is differentiable at zero, and the order of the kernel is determined by the number of derivatives of $1/(1+\lambda\alpha^2(\omega))$ which vanish at zero.

5. Extensions. The construction of α-splines can be extended to more general inverse problems.

5.1. Convolution. Quite often in experimental situations, rather than measuring the value of the unknown function at a design point, a physical device can only measure a weighted average of the function in a neighborhood of this point. This leads us to consider convolution functionals L_i of the form

$$L_i f=(w*f)(t_i)=\int_0^1 w(u)f(t_i-u)\,du \tag{49}$$

where the kernel w is square integrable on $(0,1)$. But, evaluation and differentiation can be considered of the same type if we broaden the framework and work with generalized periodic functions: evaluation is then convolution with δ_0 and differentiation is convolution by derivatives of δ_0. A trigonometric series

$$\sum_{n=-\infty}^{\infty} c_n\, e^{2\pi int} \tag{50}$$

with complex coefficients c_n is a periodic generalized function f (a periodic tempered distribution in the Schwartz sense; see [8]) if and only if the coefficients c_n are slowly increasing at ∞, i.e.,

$$\exists A\in R,\quad \exists \nu\in N:\quad |c_n|\leqq A|n|^\nu \quad \forall n. \tag{51}$$

The c_n are the generalized Fourier coefficients of f. If $\sum_{n=-\infty}^{\infty}|c_n|^2<\infty$, then (50) defines an $\mathscr{L}_2(0,1)$ function and the c_n coincide with the usual Fourier coefficients. If c_n and d_n both satisfy condition (51), it is easy to see that c_nd_n also satisfies (51) and hence $\sum_{n=-\infty}^{\infty} c_nd_n\, e^{2\pi int}$ defines a periodic generalized function, which is, by definition, the convolution product of $f(t)=\sum_{n=-\infty}^{\infty} c_n\, e^{2\pi int}$ by $g(t)=\sum_{n=-\infty}^{\infty} d_n\, e^{2\pi int}$, and which we denote by $f*g$. This definition coincides with the classical one (49) when f and g are both ordinary continuous functions in the sense that $f*g$ is equivalent to $\int_0^1 f(\cdot-s)g(s)\,ds$. Then evaluation corresponds to convolution by the periodic δ_0 function defined by $(\delta_0)_n\equiv 1$ since

$$f_n=1f_n \Rightarrow f(t)=(f*\delta_0)(t).$$

Similarly, differentiation of order k corresponds to convolution with a kernel ω defined by $\omega_n=(2\pi in)^k$.

We will consider spaces $\mathscr{E}_\alpha$ defined as in § 2.1. Remember that a requirement for the regression problem (1.1) is that the functionals L_i be continuous. Hence for the convolution problem, we do not need, in general, that evaluation be continuous (i.e., $\mathscr{E}_\alpha$ be a RKHS). This requirement will correspond to conditions involving both α and the kernel ω, which, for a given kernel, determine a range of compatible penalties (the smoother the kernel, the "larger" the spaces $\mathscr{E}_\alpha$ in this range).

Let ω be a fixed kernel (a periodic generalized function). Let α be a real-valued positive even function such that: (J-1) the sequence $1/\alpha(n)$ is bounded by $P(n)$ for some polynomial P (J-2) $\sum_k (|\omega_k|/\alpha(k))^2 \leqq \infty$. These assumptions replace (A3). (J-1) is a very mild condition preventing the weights $\alpha(n)$ from being too small, and is technically needed for the map T (see (11)) to be an isomorphism. (J-2) will guarantee continuity of the convolution functionals (see Lemma 5.1 below). In the evaluation case, (J-1) is a consequence of (J-2). Note that for smooth kernels, (J-1) and (J-2) are less restrictive than (A3).

Let $\mathscr{E}_\alpha$ be the space of periodic generalized functions such that (E) is satisfied. The arguments of Lemma 2.1 are still valid. The only difference is that now, having removed assumption (A3), continuity of the elements of $\mathscr{E}_\alpha$ is no longer a corollary of this lemma for all α. Formula (8) defines a scalar product in $\mathscr{E}_\alpha$ and the fact that this scalar product confers to $\mathscr{E}_\alpha$ a structure of Hilbert space is similar to the corresponding argument in Lemma 2.2 where we replace "$(r_n) \in l_2$" by "(r_n) slowly increasing to ∞," and "$(1/\alpha(n))$ bounded" by "$(1/\alpha(n))$ bounded by some polynomial."

LEMMA 5.1. *Under the above assumptions, for each fixed t, the convolution functional $f \to (w * f)(t)$ is a continuous linear functional.*

Proof. It is first easy to check that for a fixed $f \in \mathscr{E}_\alpha$, $t \to (w * f)(t)$ is a continuous function: this follows from the fact that $\sum (|w_k|/\alpha(k))^2 < \infty$ and $\sum |\alpha(k) f_k|^2 < \infty$ implies $\sum |w_k f_k| < \infty$. This gives a meaning to $\omega * f$ evaluated at t. To prove that $f \to (\omega * f)(t)$ is continuous, we will make explicit the Riesz representer $k_t^w(\cdot)$ of this linear functional. By arguments similar to § 2 again, it is defined by its Fourier coefficients:

$$(52)\qquad \begin{aligned} (k_t^w)_n &= w_n e^{-2\pi i n t} \quad \text{for } |n| < h, \\ (k_t^w)_n &= \frac{w_n}{\alpha_n^2} e^{-2\pi i n t} \quad \text{for } |n| \geqq h. \end{aligned} \qquad \square$$

Given noisy measurements y_i of $(w * f)(t_i)$, for $i = 1, \cdots, n$, the proposed estimate of f will minimize

$$(53)\qquad \frac{1}{n} \sum_{i=1}^{n} ((w * g)(t_i) - y_i)^2 + \sum_{|n| \geqq h} \alpha^2(n) |g_n|^2$$

over $g \in \mathscr{E}_\alpha$. For this problem to have a unique solution, we need an additional assumption on w, because if w was, for example, a trigonometric polynomial of order $q < n$, then the representers of convolution $k_{t_1}^w, \cdots, k_{t_n}^w$ would be linearly dependent. A convenient way of ruling this out is to assume, for example, that $w_k \neq 0$ for $k = 0, \cdots, n-1$.

LEMMA 5.2. *Under the above assumptions, and if $n \geqq 2h-1$, the functional* (53) *has a unique minimizer in $\mathscr{E}_\alpha$.*

Proof. The proof is similar to the proof of Lemma 2.2 with the additional arguments that if g is a trigonometric polynomial of order $\leqq h-1$, then so is $w * g$, and that if

$$\sum_{p=-\infty}^{\infty} (k_0^w)_p \left| \sum_{i=1}^{n} z_i e^{2\pi i p t} \right|^2 = 0,$$

then since $(k_0^w)_p \neq 0$ for $p=0,\cdots,n-1$, we have $\sum_{i=1}^n z_i e^{2\pi i p t_i}=0$ for $p=0,\cdots,n-1$, which implies that all the z_i are zero. □

Note that, in the case when w_p is always nonzero, the procedure amounts to computing an evaluation β-spline g, for $\beta(p)=\alpha(p)/|w_p|$ based on the data y_i, and then deconvolve g. For a prescribed smoothness on f, the procedure is adapting the suitable weight on $g=w*f$.

5.2. Multidimensional aspect. It is not difficult to generalize to periodic functions of several variables, using penalties of the form

$$J_\alpha(f)=\sum_{\nu_1=-\infty}^{\infty}\cdots\sum_{\gamma_d=-\infty}^{\infty}\alpha(\nu_1,\cdots,\nu_d)^2|f_{\nu_1\cdots\nu_d}|^2 \tag{54}$$

where

$$f_{\nu_1,\cdots,\nu_d}=\int_0^1\cdots\int_0^1 f(x_1,\cdots,x_d)\exp\left(-2\pi i\sum_{l=1}^d x_l\nu_l\right)dx_1\cdots dx_d.$$

But there is an extra condition on the design points to ensure uniqueness of the solution to the minimization problem, which is that any function with a zero penalty and vanishing on the design points must vanish everywhere.

This class of penalties includes

$$J(f)=\int_0^1\cdots\int_0^1\left(\frac{\partial^{dm}f}{\partial x_1^m\cdots\partial x_d^m}\right)^2 dx_1\cdots dx_d \tag{55}$$

with

$$\alpha^2(\nu_1,\cdots,\nu_d)=\prod_{i=1}^d(2\pi\nu_1)^{2m}$$

and also "thin plate splines" penalties:

$$J(f)=\sum_{\nu_1,\cdots,\nu_d=-\infty}^{\infty}[(2\pi\nu_1)^2+\cdots+(2\pi\nu d)^2]^m|f_{\nu_1\cdots\nu_d}|^2 \tag{56}$$

where $\alpha^2(\nu_1,\cdots,\nu_d)=\|\nu\|^{2m}$ is homogeneous in frequency space.

6. Numerical evaluation for the choice of penalty. Asymptotic theory predicts that, if the unknown function f is very smooth, then the integrated mean square error should be smaller for an exponentially growing weight function α than for a polynomially growing weight function such as the one cubic splines use. It is then interesting to see whether this shows numerically for reasonable sample sizes and, moreover, to test how sensitive the method is to the actual smoothness of the true function.

We concentrate on two one-parameter families of estimates:

(F1) g_λ^1 is the periodic α-spline for $\alpha^2(w)=\lambda\, e^{\gamma w}(h=1)$.

(F2) g_λ^2 is the cubic spline for $\alpha^2(w)=\lambda w^4(h=1)$.

We choose the parameter $\gamma=4\log 2$ so that the weights behave similarly for low frequencies.

The assessment of an estimate g_λ of the true function f is based on the discrete risk

$$R(g_\lambda)=c_n^2(\lambda)+\mu_n(\lambda) \tag{57}$$

and the criteria we use for evaluating a one-parameter family of estimates g_λ is

$$C=\min_{\lambda\in\mathbb{R}^+}R(g_\lambda).$$

Let C_1 and C_2 denote the values of C for families (F1) and (F2), respectively. We

TABLE 1

Frequency = 3	$N=50$			$N=100$		
	C_1/C_0	C_2/C_0	C_2/C_1	C_1/C_0	C_2/C_0	C_2/C_1
$S/N=100$	5.21	12.44	2.39	4.89	12.24	2.50
$S/N=20$	4.63	8.62	1.86	4.36	8.54	1.96
$S/N=5$	4.15	6.29	1.51	3.92	6.24	1.59
$S/N=1$	3.00	3.62	1.20	3.36	4.27	1.27

TABLE 2

$S/N=100$	$N=50$			$N=100$		
	C_1/C_0	C_2/C_0	C_2/C_1	C_1/C_0	C_2/C_0	C_2/C_1
$k=1$	3.86	5.48	1.42	3.60	5.34	1.48
$k=3$	5.21	12.43	2.39	4.89	12.24	2.50
$k=5$	9.13	24.82	2.72	6.85	19.89	2.90
$k=10$	12.69	25.53	2.01	11.00	35.77	3.16

measure the signal-to-noise ratio by the following quantity:

$$\frac{S}{N}=\frac{1}{\sigma}\left(\int_0^1\left(f(t)-\int_0^1 f(s)\,ds\right)^2 dt\right) \tag{58}$$

where σ is the deviation of the error distribution.

6.1. Tables 1 and 2. In this section, the true function f is of the form

$$f(t)=a\sin(2\pi kt) \tag{59}$$

where k is a positive integer, and $t\in(0,1)$. Let g^0 be the parametric estimate given by a nonlinear regression of the form (59), which is the best method we could apply in this situation, and let C_0 be $R(g^0)$. By a simulation of size 100, we get an approximation to C_0 for each of the functions considered. In Tables 1 and 2, we report the values of C_1/C_0, C_2/C_0 and the relative efficiency C_2/C_1, for a sample size of $n=50$ and $n=100$. In Table 1, the frequency k is fixed to $k=3$ and the signal-to-noise ratio takes values 100, 20, 5, and 1. In Table 2, the signal-to-noise ratio is fixed to $S/N=100$ and the frequency takes values $k=1$, 3, 5, 10.

We observe that the relative efficiency decreases toward one as σ increases: in presence of a lot of noise, all the methods (i.e., choice of weight) tend to be equivalent. The difference between polynomial weight and exponential weight gets emphasized by larger sample sizes, and also for higher frequencies.

6.2. Tables 3, 4, and 5. When we use a periodic α-spline, the true function f should satisfy (E) that, in the case of family (1) is

$$\sum_{|n|\geqq 1} e^{2\gamma|n|}|f_n|^2<\infty \tag{I}$$

and in case of family (2)

$$\sum_{|n|\geqq 1} |n|^4|f_n|^2<\infty. \tag{II}$$

TABLE 3

Function f		$N=50$			$N=100$			$N=500$		
		$(F1)$	$(F2)$	C_2/C_1	$(F1)$	$(F2)$	C_2/C_1	$(F1)$	$(F2)$	C_2/C_1
	ρ	.049	.11		.049	.11		.047	.11	
$S/N=100$	NP	6.7	9.6	1.42	6.9	10.3	1.48	7.5	12.4	1.64
	NO	7.4	5.2		14.3	9.66		66.2	40.4	
	ρ	.068	.10		.066	.11		.06	.11	
$S/N=20$	NP	5.6	6.7	1.20	5.8	7.2	1.24	6.4	8.7	1.35
	NO	8.9	7.5		17.18	13.82		77.8	57.7	
	ρ	.074	.10		.07	.10		0.69	.11	
$S/N=5$	NP	4.6	4.9	1.06	4.8	5.3	1.10	5.4	6.3	1.18
	NO	10.9	10.19		20.7	18.85		92.8	78.7	
	ρ	.11	.11		.10	.11		.093	.022	
$S/N=1$	NP	3.4	3.4	1.00	3.6	3.7	1.01	4.2	4.8	1.15
	NO	14.9	14.8		27.6	27.23		117.7	102.36	

TABLE 4

Function g		$N=50$			$N=100$			$N=500$		
		$(F1)$	$(F2)$	C_2/C_1	$(F1)$	$(F2)$	C_2/C_1	$(F1)$	$(F2)$	C_2/C_1
	ρ	.15	.16		.55	.27		.73	.25	
$S/N=100$	NP	35.5	32.8	.92	78.1	56.7	.72	144.3	75.6	.52
	NO	1.4	1.52		1.28	1.76		3.46	6.6	
	ρ	.26	.21		.31	.27		.32	.25	
$S/N=20$	NP	19.2	17.5	.91	26.1	23.1	.89	38.5	33.2	.86
	NO	2.6	2.9		3.8	4.3		12.9	15.0	
	ρ	.31	.24		.21	.26		.24	.25	
$S/N=5$	NP	9.5	9.1	.97	12.1	11.2	.92	18.2	16.6	.90
	NO	5.2	5.5		8.24	8.93		27.4	30.2	
	ρ	.24	.23		.33	.28		.22	.25	
$S/N=1$	NP	4.0	3.9	.98	4.9	4.7	.96	7.6	7.4	.98
	NO	12.4	12.6		20.2	21.0		65.8	67.3	

We choose three functions $f(\cdot)$, $g(\cdot)$, $h(\cdot)$, with the same signal-to-noise ratio, so that:

(1) $f(\cdot)$ satisfies (I) and (II).

(2) $g(\cdot)$ satisfies neither (I) nor (II).

(3) $h(\cdot)$ satisfies (II) but not (I).

They are given by

$$\begin{aligned} f(x) &= \sqrt{2}\sin(2\pi x) && x \in [0, 1], \\ g(x) &= 4\sqrt{3}x && x \in \left[0, \frac{1}{4}\right] \\ &= 4\sqrt{3}\left(\frac{1}{2} - x\right) && x \in \left[\frac{1}{4}, \frac{3}{4}\right] \\ &= 4\sqrt{3}(x-1) && x \in \left[\frac{3}{4}, 1\right], \end{aligned}$$

TABLE 5

Function h		$N=50$			$N=100$			$N=500$		
		$(F1)$	$(F2)$	C_2/C_1	$(F1)$	$(F2)$	C_2/C_1	$(F1)$	$(F2)$	C_2/C_1
	ρ	.13	.16		.20	.15		.26	.16	
$S/N=100$	NP	20.2	20.3	1.00	22.9	22.7	.99	31.5	29.4	.93
	NO	2.5	2.5		4.4	4.4		15.8	16.9	
	ρ	.14	.15		.19	.15		.17	.16	
$S/N=20$	NP	11.7	12.5	1.07	13.1	13.9	1.06	17.3	17.8	1.03
	NO	4.27	3.4		7.6	7.2		28.9	28.1	
	ρ	.12	.16		.16	.15		.12	.15	
$S/N=5$	NP	7.6	8.3	1.09	8.4	9.2	1.10	11.0	11.7	1.06
	NO	6.6	6.0		11.9	10.8		45.3	42.7	
	ρ	.31	.28		.40	.33		.09	.17	
$S/N=1$	NP	4.0	3.9	.98	5.1	4.8	.95	6.9	7.2	1.03
	NO	12.4	12.7		19.4	20.5		72.5	69.7	

$$\begin{aligned} h(x) &= \frac{2}{\sqrt{3}}\sin^2(2\pi x) \qquad x\in\left[0,\frac{1}{2}\right] \\ &= \frac{-2}{\sqrt{3}}\sin^2(2\pi x) \qquad x\in\left[\frac{1}{2},1\right]. \end{aligned}$$

Since the value of the criteria C depends on the sample size, the variance of the noise, etc., we think it is quite difficult to visualize how good a procedure is by just looking at C. For this reason, we report instead the following two quantities:

$$\text{NP}=\frac{nC}{\sigma^2}, \tag{60}$$

$$\text{NO}=\frac{\sigma^2}{C}. \tag{61}$$

To interpret NP and NO, remember that for a parametric least squares estimate, using p parameters, the variance component of the risk would be $\sigma^2 p/n$. Hence we can think of NP as being roughly the "number of parameters" that a parametric method with the same risk would use (assuming it would have no bias). NP is a convenient multiple of C. For the same reason, we can think of NO as being the number of observations used per parameter estimated.

In Tables 3, 4, and 5, we report the values of NP, NO, and C_2/C_1 for function f (respectively, g and h), for sample sizes $n=50$, 100, and 500, and the same choices of signal-to-noise ratio as previously. We also report the ratio ρ of bias-to-risk in the risk corresponding to the optimal choice of λ.

We observe again that, as expected, the difference between exponential and polynomial weight measured by C_2/C_1, increases with the sample size and with the signal-to-noise ratio. For function f, the most favorable case for exponential weight, C_2/C_1 goes as high as 1.64 for a sample size of 500 and a signal-to-noise ratio of 100. Even though function h violates assumption (I), we see that C_2/C_1 is very close to one in most cases, and the lowest it gets is .93. This seems to indicate that overestimating a priori the smoothness of the function does not hurt too much. For function g, polynomial weights perform better as expected, and C_2/C_1 gets as low as .52 for $n = 500$ and $S/N = 100$, but is still fairly close to one for small sample size.

Another important issue in nonparametric regression is the choice of the smoothing parameter. Two important methods for that purpose are that of Mallows [7] and the generalized cross validation method (Golub, Heath, and Wahba [4]). The asymptotic optimality of these two methods when using D^m-splines is established by Li [6]. It is easy to see that this can be extended to exponential weights. The argument relies on the fact that the coefficient of variation of the eigenvalues of the information matrix (reciprocals of eigenvalues of Ω) converges to zero when n converges to infinity.

Acknowledgments. I thank my advisor Ker Chau Li for his advice and support during this work, and the referees for their very useful comments.

REFERENCES

[1] P. M. Anselone and P. J. Laurent, *A general method for the construction of interpolating or smoothing spline functions*, Numer. Math., 12 (1968), pp. 66-82.

[2] R. Cogburn and H. T. Davis, *Periodic splines and spectral estimation*, Ann. Statist., 2 (1974), pp. 1108-1126.

[3] R. L. Eubank, *Spline Smoothing and Nonparametric Regression*, Marcel Dekker, New York, 1988.

[4] G. H. Golub, M. Heath, and G. Wahba, GCV *as a method for choosing a good ridge parameter*, Technometrics, 21 (1979), pp. 215-222.

[5] G. Kimeldorf and G. Wahba, *Some results on Tchebicheffian spline functions*, J. Math. Anal. Appl., 33 (1971), pp. 82-95.

[6] K. C. Li, *Asymptotic optimality of C_L and generalized cross validation in ridge regression with applications to spline smoothing*, Ann. Statist., 14 (1986), pp. 1101-1112.

[7] C. L. Mallows, *Some comments on C_p*, Technometrics, 15 (1973), pp. 661-675.

[8] L. Schwartz, *Theory of Distributions*, Paris, 1950.

[9] B. Silverman, *Spline smoothing: the equivalent variable kernel method*, Ann. Statist., 12 (1984), pp. 898-916.

[10] P. Speckman, *The asymptotic integrated mean square error for smoothing noisy data by splines*, Numer. Math., to appear.

[11] ———, *Spline smoothing and optimal rates of convergence in non-parametric regression models*, Ann. Statist., 13 (1985), pp. 970-983.

[12] G. Wahba and S. Wold, *Periodic splines for spectral density estimation: the use of cross validation for determining the degree of smoothing*, Comm. Statist., 4 (1975), pp. 125-141.

[13] G. Wahba, *Smoothing noisy data with spline functions*, Numer. Math., 24 (1975), pp. 383-393.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 3, pp. 503-518, May 1990

TRUNCATED SINGULAR VALUE DECOMPOSITION SOLUTIONS TO DISCRETE ILL-POSED PROBLEMS WITH ILL-DETERMINED NUMERICAL RANK*

PER CHRISTIAN HANSEN†

Abstract. Tikhonov regularization is a standard method for obtaining smooth solutions to discrete ill-posed problems. A more recent method, based on the singular value decomposition (SVD), is the truncated SVD method. The purpose of this paper is to show, under mild conditions, that the success of both truncated SVD and Tikhonov regularization depends on satisfaction of a discrete Picard condition, involving both the matrix and the right-hand side. When this condition is satisfied, then both methods are guaranteed to produce smooth solutions that are very similar.

Key words. ill-posed problems, truncated SVD, regularization in standard form, perturbation theory

AMS(MOS) subject classifications. 65F20, 65R20

1. Introduction. This paper is concerned with the linear least squares problem:

$$\min \|Ax-\mathbf{b}\|_2, \qquad A\in\mathbb{R}^{m\times n}, \qquad m\geqq n, \tag{1}$$

where the matrix A is ill-conditioned and has *ill-determined numerical rank* (i.e., its singular values decay gradually towards zero without any particular gap in the spectrum). Such problems typically arise in connection with the numerical solution of Fredholm integral equations of the first kind,

$$\int K(s,x)f(x)\,dx=g(s), \tag{2}$$

which are classical examples of ill-posed problems. The following discussion is particularly focused on ill-conditioned least squares problems arising from the discretization of (2), but we stress that the results hold for discrete ill-posed problems in general. Throughout the paper, we shall assume for the sake of simplicity that the matrix A has full rank.

The singular value decomposition (SVD) is an invaluable tool for analysis of problems with ill-conditioned matrices, and the truncated SVD (TSVD) method has been used successfully to solve a variety of discrete ill-posed problems of the form (1). In spite of this, the method still lacks some theoretical background. Our aim here is to develop a theory for the TSVD and thus provide insight into its behavior. To do this, we find it useful to compare the method with another widely used method for ill-posed problems, namely, Tikhonov regularization. This method is theoretically better understood than the TSVD, but there seems to be no general criteria by which these methods can be compared [14]. In [9], Hansen showed that if there is a distinct gap in the singular value spectrum, then TSVD is equivalent to Tikhonov regularization. The present work continues this investigation, with attention primarily focused on matrices with ill-determined numerical rank. We show that the existence of a satisfactory

* Received by the editors February 3, 1988; accepted for publication (in revised form) February 9, 1989. Part of this work was carried out while the author was visiting the University of California at Los Angeles (UCLA) and was supported by the Danish Natural Science Foundation, by the National Science Foundation under contract NSF-DMS87-14612, and by the Army Research Office under contract DAAL03-88-K-0085.

† Danish Computing Centre for Research and Education (UNI • C), Building 305, Technical University of Denmark, DK-2800 Lyngby, Denmark.

approximate solution primarily depends on satisfaction of a discrete Picard condition and in fact (even for matrices with well-determined numerical rank) has little to do with finding the numerical rank of a matrix. We also show that once the discrete Picard condition is satisfied, then TSVD and Tikhonov regularization always yield very similar solutions. The work was inspired by the "trilogy" of papers by Varah [22]-[24] and the paper by Aulick and Gallie [1].

The paper is organized as follows. In § 2 we introduce the methods of truncated SVD and Tikhonov regularization. In § 3 we show that the convergence of both methods largely depends on the behavior of the right-hand side, and we formulate the discrete Picard condition. Section 4 presents perturbation bounds for the methods, and in § 5 we further characterize the behavior of the solutions under the influence of errors. Finally, in § 6 we give two numerical examples.

2. Truncated SVD and Tikhonov regularization. Our investigation takes its basis in the *singular value expansion* (SVE), which is a mean convergent expansion of the kernel K in the form

$$K(s,x)=\sum_{i=1}^{\infty}\mu_i u_i(s)v_i(x),$$

where both $\{u_i\}$ and $\{v_i\}$ are sequences of orthonormal functions, and all $\mu_i \geqq 0$. In terms of the SVE, the solution f to (2) can be written as

$$f(x)=\sum_{i=1}^{\infty}\frac{(u_i,g)}{\mu_i}v_i(x), \tag{3}$$

where (u_i, g) denotes the usual inner product. See, e.g., [8, § 1.2] for more details. The ill-posed nature of (2) is reflected in the facts that the sequence $\{\mu_i\}$ has zero as its only limit point, and the "smoother" the kernel the faster the μ_i decay to zero [5, Thm. 3.2]. Hence, a square integrable solution f can exist only if the coefficients (u_i, g) decay to zero faster than the μ_i, such that

$$\sum_{i=1}^{\infty}\left(\frac{(u_i,g)}{\mu_i}\right)^2<\infty. \tag{4}$$

This is the well-known *Picard condition* [8, Thm. 1.2.6].

Corresponding to the SVE of K, the matrix A has a *singular value decomposition* (SVD) in the form [2, § 3]

$$A=U\Sigma V^T=\sum_{i=1}^{n}\sigma_i \mathbf{u}_i\mathbf{v}_i^T,$$

where the left and right singular vectors $\mathbf{u}_i$ and $\mathbf{v}_i$ are the orthonormal columns of the matrices $U\in\mathbb{R}^{m\times n}$ and $V\in\mathbb{R}^{n\times n}$, and the singular values σ_i are the diagonal elements of $\Sigma\in\mathbb{R}^{n\times n}$. They satisfy $\sigma_1\geqq\sigma_2\geqq\cdots\geqq\sigma_n$ and, since A is assumed to have full rank, $\sigma_n>0$. The relationship between the ill-posedness of (2) and the large condition number σ_1/σ_n of the matrix A in (1) was studied by Richter [17] and Wing [25]. Recently, Hansen [10] elaborated on this by showing that whenever (2) is discretized by an expansion method with orthonormal basis functions, the SVD of A is closely related to the SVE of K in the sense that the σ_i and $\mathbf{u}_i^T\mathbf{b}$ are approximations to the μ_i and (u_i, g), respectively. This means that if g satisfies the Picard condition (4) and is unaffected by errors, and if the order n of the discretization is sufficiently large, then the *exact least squares solution* $\mathbf{x}_o$ to the unperturbed discretized problem (1), given by

$$\mathbf{x}_o \equiv A^+\mathbf{b} = \sum_{i=1}^{n} \frac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i}\mathbf{v}_i, \tag{5}$$

yields an approximation to the solution f given in (3). See also [8, Thm. 4.1.6].

When g and, equivalently, $\mathbf{b}$ are perturbed by errors, then the solution to the perturbed problem is very likely to be dominated by errors that are "blown up" by the small singular values μ_i or σ_i in the denominators of (3) or (5). It is therefore necessary to apply some sort of regularization to either (2) or (1) to compute a solution that is less sensitive to the perturbations and that approximates the exact solution to the unperturbed problem, f (3) or $\mathbf{x}_o$ (5). Due to the close relationship between the SVD and the SVE, applying a certain regularization method to (1) is equivalent to applying the same regularization method to (2) [10], and the convergence of the regularized algebraic solutions to (1) carries over to the corresponding approximate solutions to (2) [8, § 4.2].

A highly regarded regularization method, due to Tikhonov [20], amounts to defining the *regularized solution* $\mathbf{x}_\lambda$ as the unique solution to the following least squares problem with a quadratic constraint:

$$\min\{\|A\mathbf{x}-\mathbf{b}\|_2^2+\lambda^2\|\mathbf{x}\|_2^2\}. \tag{6}$$

Here, the *regularization parameter* λ controls the "smoothness" of the regularized solution. We remind the reader that $\mathbf{x}_\lambda$ can always be written in terms of the SVD as

$$\mathbf{x}_\lambda = \sum_{i=1}^{n} \frac{\sigma_i^2}{\sigma_i^2+\lambda^2}\frac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i}\mathbf{v}_i. \tag{7}$$

If we compare this equation with (5), we see that the role of the regularization parameter λ is to dampen or filter the terms in the sum corresponding to singular values smaller than about λ. Hence, in any practical application, λ will always satisfy $\sigma_n \leqq \lambda \leqq \sigma_1$.

An alternative method for regularization of (1) is the *truncated* SVD (TSVD) method, in which we discard the smallest singular values simply by truncating the sum in (5) at some $k<n$ [22]. Thus, the TSVD *solution* $\mathbf{x}_k$ is defined by

$$\mathbf{x}_k \equiv \sum_{i=1}^{k} \frac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i}\mathbf{v}_i = U\Sigma_k^+V^T\mathbf{b}, \qquad \Sigma_k^+ \equiv \operatorname{diag}(\sigma_1^{-1},\cdots,\sigma_k^{-1},0,\cdots,0). \tag{8}$$

The integer k is called the *truncation parameter*, and it plays a role similar to the λ in (7). Note that $\mathbf{x}_o$ is identical to $\mathbf{x}_\lambda$ with $\lambda=0$ and $\mathbf{x}_k$ with $k=n$. We stress that the TSVD solution can be computed at least as efficiently as the regularized solution, without the large computational effort involved in a complete SVD computation (cf. the survey of methods in [11]). For more details, computational aspects, and examples of the application of these methods, see, e.g., [2], [3], [5], [6], [8], [22]-[24].

The use of Tikhonov regularization as well as TSVD is based on the following heuristic:

HEURISTIC 2.1. *The number of oscillations in the left and right singular vectors* $\mathbf{u}_i$ *and* $\mathbf{v}_i$ *tends to increase with increasing i.*

When this is true (which is the case, e.g., if the matrix A is totally positive), it is obvious that the TSVD solution $\mathbf{x}_k$ as well as the regularized solution $\mathbf{x}_\lambda$ tend to be smoother than the least squares solution $\mathbf{x}_o$. We are aware that in some applications Heuristic 2.1 is not satisfied or $\mathbf{x}_o$ is simply not the solution we are interested in (even without noise being present) because the sought solution f does not have a nice representation in terms of the right singular functions v_i. In these cases, we should replace the term $\|\mathbf{x}\|_2$ in (6) by another appropriate regularization term such as, e.g., $\|L\mathbf{x}\|_2$, as pointed out in [23]. We are also aware that this corresponds to an expansion of the solution in terms of the generalized SVD of the matrix pair (A, L) [12], [23]. However, we feel that a fundamental understanding of the simpler case (6), as provided in this paper, is necessary before we can proceed to perform an analysis of the general case.

3. The convergence of the methods. Before starting our discussion, we make the assumption that k and λ are chosen such that the solutions $\mathbf{x}_\lambda$ and $\mathbf{x}_k$ are not too different—otherwise, there will be no point in making a comparison between them. From the expression (7) for $\mathbf{x}_\lambda$ we see that this is the case where $\lambda \approx \sigma_k$, since then the damping of the terms in (7) sets in for singular values smaller than about σ_k. It can actually be proved [9, Thm. 5.2] that λ chosen somewhere in the range $(\sigma_k^3\sigma_{k+1})^{1/4} \leqq \lambda \leqq (\sigma_k\sigma_{k+1})^{1/2}$ brings $\mathbf{x}_k$ and $\mathbf{x}_\lambda$ as close as possible. In § 5 we return to the actual choice of k and λ.

The main goal of this section is to show how the behavior of the right-hand side $\mathbf{b}$ in (1) influences the convergence of the TSVD solution $\mathbf{x}_k$ and the regularized solution $\mathbf{x}_\lambda$. For this purpose we set up, in terms of the SVD of A, the following "model" of a right-hand side:

$$\beta_i = \mathbf{u}_i^T\mathbf{b} = \sigma_i^\alpha, \qquad i = 1, \cdots, n, \qquad \alpha \geqq 0, \tag{9}$$

where the nonnegative real constant α determines the decay of the β_i relative to the σ_i (when $\alpha > 1$, the β_i decay faster than the σ_i). According to (9), we can write $\mathbf{b} = U\Sigma^\alpha\mathbf{f}$ with $\mathbf{f} = [1, 1, \cdots, 1]^T$. Although (9) is a crude "model" of the right-hand sides as they appear in practical applications, it is sufficiently realistic to clearly illustrate the importance of the decay of the β_i—rather than the particular shape of the singular values spectrum of A.

First, we investigate the convergence of the solutions $\mathbf{x}_k$ and $\mathbf{x}_\lambda$; i.e., we shall determine how well they approximate the exact least squares solution $\mathbf{x}_o$ (5) to the unperturbed problem. Following [1], the differences $\mathbf{x}_o - \mathbf{x}_k$ and $\mathbf{x}_o - \mathbf{x}_\lambda$ are called the TSVD *error* and the *regularization error*, respectively. The closeness of $\mathbf{x}_k$ and $\mathbf{x}_\lambda$ to $\mathbf{x}_o$ is illustrated in the following theorem.

THEOREM 3.1. *Let $\mathbf{x}_k$ and $\mathbf{x}_\lambda$ denote the solutions* (8) *and* (7), *and let $\mathbf{x}_o$ denote the exact least squares solution* (5). *Further, let the right-hand side* $\mathbf{b}$ *satisfy* (9). *Then the norms of the* TSVD *error $\mathbf{x}_o - \mathbf{x}_k$ and the regularization error $\mathbf{x}_o - \mathbf{x}_\lambda$ satisfy*

$$\frac{\|\mathbf{x}_o - \mathbf{x}_k\|_2}{\|\mathbf{x}_o\|_2} \leqq \begin{cases} \sqrt{n}, & 0 \leqq \alpha < 1 \\ (\sigma_{k+1}/\sigma_1)^{\alpha-1}\sqrt{n}, & 1 \leqq \alpha \end{cases} \tag{10a}$$

$$\frac{\|\mathbf{x}_o - \mathbf{x}_\lambda\|_2}{\|\mathbf{x}_o\|_2} \leqq \begin{cases} \sqrt{n}, & 0 \leqq \alpha < 1 \\ (\lambda/\sigma_1)^{\alpha-1}\sqrt{n}, & 1 \leqq \alpha < 3 \\ (\lambda/\sigma_1)^2\sqrt{n}, & 3 \leqq \alpha. \end{cases} \tag{10b}$$

Proof. To simplify the notation, we introduce the quantities

$$\boldsymbol{\xi}_o = V^T\mathbf{x}_o, \qquad \boldsymbol{\xi}_k = V^T\mathbf{x}_k, \qquad \boldsymbol{\xi}_\lambda = V^T\mathbf{x}_\lambda. \tag{11}$$

We shall first derive a lower bound on $\|\mathbf{x}_o\|_2$. According to (5),

$$\mathbf{x}_o = A^+\mathbf{b} = V\Sigma^{-1}U^TU\Sigma^\alpha\mathbf{f} = V\Sigma^{\alpha-1}\mathbf{f} = V[\sigma_1^{\alpha-1}, \cdots, \sigma_n^{\alpha-1}]^T \Leftrightarrow$$

$$\|\mathbf{x}_o\|_2 = \|\boldsymbol{\xi}_o\|_2 \geqq \max_i\{\sigma_i^{\alpha-1}\} = \begin{cases} \sigma_n^{\alpha-1}, & 0 \leqq \alpha < 1 \\ \sigma_1^{\alpha-1}, & 1 \leqq \alpha. \end{cases}$$

Concerning the norm of the TSVD error, we have

$$\|\mathbf{x}_o-\mathbf{x}_k\|_2=\|\boldsymbol{\xi}_o-\boldsymbol{\xi}_k\|_2\leqq\sqrt{n}\|\boldsymbol{\xi}_o-\boldsymbol{\xi}_k\|_\infty=\sqrt{n}\|(\Sigma^+-\Sigma_k^+)\Sigma^\alpha\mathbf{f}\|_\infty$$

$$=\max_{k+1\leqq i\leqq n}\{\sigma_i^{\alpha-1}\}\cdot\sqrt{n}=\begin{cases}\sigma_n^{\alpha-1}\sqrt{n}, & 0\leqq\alpha<1\\ \sigma_{k+1}^{\alpha-1}\sqrt{n}, & 1\leqq\alpha.\end{cases}$$

These relations immediately lead to (10a). To obtain a bound on the norm of the regularization error, we have, in analogy with above, $\|\mathbf{x}_o-\mathbf{x}_\lambda\|_2\leqq\sqrt{n}\|\mathrm{diag}\,(\lambda^2/(\sigma_i^2+\lambda^2))\Sigma^{\alpha-1}\mathbf{f}\|_\infty=\lambda^2\max\{\sigma_i^{\alpha-1}/(\sigma_i^2+\lambda^2)\}$. Define the function $\phi(\sigma)=\sigma^{\alpha-1}/(\sigma^2+\lambda^2)$. For $0\leqq\alpha<1$, ϕ is a decreasing function with maximum attained for $\sigma=\sigma_n$, implying that $\|\mathbf{x}_o-\mathbf{x}_\lambda\|_2\leqq\lambda^2\sigma_n^{\alpha-1}/(\sigma_n^2+\lambda^2)\leqq\sigma_n^{\alpha-1}$. Similarly, for $\alpha\geqq3$, ϕ is an increasing function with maximum attained for $\sigma=\sigma_1$, implying that $\|\mathbf{x}_o-\mathbf{x}_\lambda\|_2\leqq\lambda^2\sigma_1^{\alpha-1}/(\sigma_1^2+\lambda^2)\leqq\lambda^2\sigma_1^{\alpha-1}/\sigma_1^2=\lambda^2\sigma_1^{\alpha-3}$. Finally, for $1\leqq\alpha<3$, ϕ attains its maximum for $\sigma=\bar{\sigma}=\lambda^2(\alpha-1)(3-\alpha)$, and

$$\|\mathbf{x}_o-\mathbf{x}_\lambda\|_2\leqq\lambda^2\bar{\sigma}^{\alpha-1}/(\bar{\sigma}^2+\lambda^2)=\tfrac{1}{2}(\alpha-1)^{1/2(\alpha-1)}(3-\alpha)^{1/2(3-\alpha)}\lambda^{\alpha-1}\leqq\lambda^{\alpha-1}.$$

These relations, together with the bound for $\|\mathbf{x}_o\|_2$, yield (10b). □

Remark. The results for $0<\alpha\leqq3$ in (10b) could also be proved using the technique from [8, Chap. 2]; but this theory does not hold for $\alpha>3$.

Theorem 3.1 shows that as long as α is larger than 1, and provided that σ_k and λ are small compared to $\sigma_1=\|A\|_2$, both the TSVD solution $\mathbf{x}_k$ and the regularized solution $\mathbf{x}_\lambda$ are *guaranteed* to approximate $\mathbf{x}_o$, and the larger α the better approximation. Usually, the truncation parameter k and the regularization parameter λ are determined by the errors in (1) in such a way that larger errors lead to a smaller k and a larger λ. Hence, Theorem 3.1 shows that if errors are present in (1), then $\mathbf{x}_k$ and $\mathbf{x}_\lambda$ can only yield satisfactory approximations to $\mathbf{x}_o$ if the coefficients β_i of the *unperturbed* right-hand side $\mathbf{b}$ decay to zero somewhat faster than the singular values σ_i. And the larger the errors, the faster the decay must be to ensure convergence.

Next, let us consider the similarity between the TSVD solution and the regularized solution by considering their difference $\mathbf{x}_k-\mathbf{x}_\lambda$ and also the difference between their residuals $(\mathbf{b}-A\mathbf{x}_k)-(\mathbf{b}-A\mathbf{x}_\lambda)=-A(\mathbf{x}_k-\mathbf{x}_\lambda)$. We assume that $\sigma_{k+1}\leqq\lambda\leqq\sigma_k$ and that the right-hand side $\mathbf{b}$ satisfies the "model" (9). A convenient way to measure the difference between $\mathbf{x}_k$ and $\mathbf{x}_\lambda$, as a function of α, is to define the following relative difference function:

$$\delta_k(\alpha)\equiv\min_{\sigma_{k+1}\leqq\lambda\leqq\sigma_k}\|\mathbf{x}_k-\mathbf{x}_\lambda\|_2\cdot\frac{\|A\|_2}{\|U^T\mathbf{b}\|_\infty}. \tag{12a}$$

Similarly, we can measure the difference between the residuals by means of the function

$$\rho_k(\alpha)\equiv\min_{\sigma_{k+1}\leqq\lambda\leqq\sigma_k}\|A(\mathbf{x}_k-\mathbf{x}_\lambda)\|_2/\|U^T\mathbf{b}\|_\infty. \tag{12b}$$

The distance function $\delta_k(\alpha)$ was briefly analyzed in [9, Thm. 6.1], and the analysis is extended in the following theorem.

THEOREM 3.2. *Let* $\mathbf{x}_k$ *and* $\mathbf{x}_\lambda$ *denote the solutions* (8) *and* (7), *and let the right-hand side* $\mathbf{b}$ *satisfy* (9). *Then upper bounds for the functions* $\delta_k(\alpha)$ *and* $\rho_k(\alpha)$ *defined in* (12a) *and* (12b) *are given by*

$$\delta_k(\alpha)\leqq\begin{cases}\sqrt{n}\cdot(\sigma_k/\sigma_1)^{\alpha-1}, & \alpha<3\\ \sqrt{n}\cdot(\sigma_k/\sigma_1)^2, & \alpha\geqq3\end{cases} \tag{13a}$$

(13b)
$$\rho_k(\alpha) \leqq \begin{cases} \sqrt{n} \cdot (\sigma_k/\sigma_1)^\alpha, & \alpha < 2 \\ \sqrt{n} \cdot (\sigma_k/\sigma_1)^2, & \alpha \geqq 2. \end{cases}$$

Proof. First, we notice that $\|\mathbf{x}_k - \mathbf{x}_\lambda\|_2 = \|\boldsymbol{\xi}_k - \boldsymbol{\xi}_\lambda\|_2 \leqq \sqrt{n}\|\boldsymbol{\xi}_k - \boldsymbol{\xi}_\lambda\|_\infty$, where $\boldsymbol{\xi}_k$ and $\boldsymbol{\xi}_\lambda$ are defined in (11). From [9, Thm. 6.1] (in particular, the third column of Table 1) it follows that

$$\delta_k(\alpha) \leqq (\sigma_k/\sigma_1)^{\alpha-1}(\sigma_k/\sigma_k)^{1/2(3-\alpha)} = (\sigma_k/\sigma_1)^{\alpha-1} \quad \text{for} \quad 0 \leqq \alpha \leqq 2,$$
$$\delta_k(\alpha) \leqq (\sigma_k/\sigma_1)^2[1+(\sigma_k/\sigma_k)^2]^{-1} \leqq (\sigma_k/\sigma_1)^2 \quad \text{for} \quad \alpha \geqq 3.$$

For α in the interval $[2, 3]$, a careful analysis of the norm

$$\|\boldsymbol{\xi}_k - \boldsymbol{\xi}_\lambda\|_\infty = \max\left\{\frac{\sigma_1^{\alpha-1}\lambda^2}{\sigma_1^2+\lambda^2}, \ldots, \frac{\sigma_k^{\alpha-1}\lambda^2}{\sigma_k^2+\lambda^2}, \frac{\sigma_{k+1}^{\alpha+1}}{\sigma_{k+1}^2+\lambda^2}, \ldots, \frac{\sigma_n^{\alpha+1}}{\sigma_n^2+\lambda^2}\right\}$$

shows that its maximum value, for $\sigma_{k+1} \leqq \lambda \leqq \sigma_k$, is given by

(14)
$$\|\boldsymbol{\xi}_k - \boldsymbol{\xi}_\lambda\|_\infty = \begin{cases} F_\alpha \cdot \lambda^{\alpha-1}, & \sigma_{k+1} \leqq \lambda \leqq \left(\dfrac{3-\alpha}{\alpha-1}\right)^{1/2}\sigma_1 \\ \dfrac{\sigma_1^{\alpha-1}\lambda^2}{\sigma_1^2+\lambda^2}, & \left(\dfrac{3-\alpha}{\alpha-1}\right)^{1/2}\sigma_1 \leqq \lambda \leqq \sigma_k \end{cases}$$

where $F_\alpha \equiv \frac{1}{2}(\alpha-1)^{1/2(\alpha-1)}(3-\alpha)^{1/2(3-\alpha)}$. Since $\lambda^2/(\sigma_1^2+\lambda^2)$ increases with λ, it follows that the minimum of (14) for $\sigma_{k+1} \leqq \lambda \leqq \sigma_k$ satisfies

$$\min_{\sigma_{k+1} \leqq \lambda \leqq \sigma_k} \|\boldsymbol{\xi}_k - \boldsymbol{\xi}_\lambda\|_\infty = \begin{cases} F_\alpha \cdot \sigma_k^{\alpha-1}, & \sigma_{k+1} \leqq \lambda \leqq \sigma_1\left(\dfrac{3-\alpha}{\alpha-1}\right)^{1/2} \\ \dfrac{\sigma_1^{\alpha-1}\sigma_k^2}{\sigma_1^2+\sigma_k^2}, & \sigma_1\left(\dfrac{3-\alpha}{\alpha-1}\right)^{1/2} \leqq \lambda \leqq \sigma_k. \end{cases}$$

Inserting this result into the expression (12a) for $\delta_k(\alpha)$, and noting that $\|A\|_2/\|U^T\mathbf{b}\|_\infty = \sigma_1^{1-\alpha}$ for all $\alpha \geqq 0$, we obtain

$$\delta_k(\alpha) = \begin{cases} F_\alpha \cdot (\sigma_k/\sigma_1)^{\alpha-1}, & \sigma_{k+1} \leqq \lambda \leqq \sigma_1\left(\dfrac{3-\alpha}{\alpha-1}\right)^{1/2} \\ (\sigma_k/\sigma_1)^2[1+(\sigma_k/\sigma_1)^2]^{-1}, & \sigma_1\left(\dfrac{3-\alpha}{\alpha-1}\right)^{1/2} \leqq \lambda \leqq \sigma_k. \end{cases}$$

Here, $F_\alpha \leqq 1$ and $[1+(\sigma_k/\sigma_1)^2]^{-1} \leqq 1$, and since $(\sigma_k/\sigma_1)^{\alpha-1} > (\sigma_k/\sigma_1)^2$ for $2 \leqq \alpha \leqq 3$, an upper bound for $\delta_k(\alpha)$ is $(\sigma_k/\sigma_1)^{\alpha-1}$. This establishes (13a). To prove (13b), we note that $\|U^T\mathbf{b}\|_\infty^{-1} = \sigma_1^{-\alpha}$ and that $\|A(\mathbf{x}_k - \mathbf{x}_\lambda)\|_2 = \|U^TA(\mathbf{x}_k - \mathbf{x}_\lambda)\|_2 = \|\Sigma(\boldsymbol{\xi}_k - \boldsymbol{\xi}_\lambda)\|_2 \leqq \sqrt{n}\|\Sigma(\boldsymbol{\xi}_k - \boldsymbol{\xi}_\lambda)\|_\infty$. Hence, $\rho_k(\alpha) = \delta_k(\alpha-1)$, and (13b) therefore follows from (13a) with α replaced by $\alpha+1$. □

Theorem 3.2 shows that when k is fixed by the errors present in (1) and σ_{k+1} is small compared to σ_1, there exists a $\lambda \in [\sigma_{k+1}, \sigma_k]$ such that $\mathbf{x}_k$ and $\mathbf{x}_\lambda$, as well as the corresponding residuals, are *guaranteed* to be close whenever α is larger than 1. And the larger α the closer the solutions and the residuals. This means that whenever the coefficients β_i of the *unperturbed* right-hand side $\mathbf{b}$ decay to zero somewhat faster than the singular values σ_i, then TSVD and Tikhonov regularization will produce approximately the same solutions and residuals, and according to Theorem 3.1 both of the solutions will approximate the unperturbed least squares solution $\mathbf{x}_o$.

We conclude this section by giving a more rigorous definition of the requirement on $\mathbf{b}$. Of course, the decay of the β_i-coefficients need not be monotonic, as long as the β_i *in average* decay to zero faster than the σ_i. We can formulate this requirement as follows.

DEFINITION 3.3. The discrete Picard condition (DPC). Let $\mathbf{b}$ denote an unperturbed right-hand side in (1). Then $\mathbf{b}$ satisfies the DPC if, for all numerically nonzero singular values σ_i, the coefficients $|\mathbf{u}_i^T\mathbf{b}|$ in average decay to zero faster than the σ_i.

We remark that if the discrete problem (1) is obtained from the integral equation (2) by means of an expansion method with orthonormal basis functions, and if the integral equation satisfies the Picard condition (4), then the DPC is also satisfied due to the relationship between the SVE and the SVD [10].

It should be stressed that while the *convergence* of $\mathbf{x}_k$ and $\mathbf{x}_\lambda$ depends on the DPC, the *smoothness* of these solutions depends on Heuristic 2.1. Thus, both vectors $\mathbf{x}_k$ and $\mathbf{x}_\lambda$ may be smooth even if the DPC is not satisfied, but in that case they will not approximate the vector $\mathbf{x}_o$. Although such solutions may still be acceptable in certain cases, it would be more correct to use the general formulation of regularization as mentioned in the last paragraph of § 2.

4. Perturbation bounds and condition numbers. In many discrete ill-posed problems, the errors are restricted to the right-hand side only. For example, this is the case if (1) is derived from an integral equation (2) whose kernel K is given exactly, e.g., from some mathematical model of a physical problem, whereas the right-hand side consists of measured quantities contaminated by errors. These errors transform directly into a perturbation of the right-hand side $\mathbf{b}$ in (1). Examples of such problems are inverse problems in observational astronomy [3], the inverse problem of electrocardiography [4], deconvolution problems such as inverse Radon and inverse Laplace transforms [15], [24], and inverse problems in computational physics (see [21] for an overview and [6] for a specific example). It is therefore appropriate to derive bounds on the perturbations of $\mathbf{x}_k$ and $\mathbf{x}_\lambda$ solely due to a perturbation $\mathbf{e}$ of $\mathbf{b}$.

THEOREM 4.1. *Let $\mathbf{x}_k$ and $\mathbf{x}_\lambda$ denote the solutions* (8) *and* (7), *and let $\tilde{\mathbf{x}}_k$ and $\tilde{\mathbf{x}}_\lambda$ denote the solutions when the right-hand side $\mathbf{b}$ of* (1) *is perturbed by $\mathbf{e}$. Assuming that $\sigma_n \leqq \lambda \leqq \sigma_1$, the relative perturbations are bounded as*

$$\frac{\|\mathbf{x}_k-\tilde{\mathbf{x}}_k\|_2}{\|\mathbf{x}_k\|_2} \leqq \frac{\sigma_1}{\sigma_k}\frac{\|\mathbf{e}\|_2}{\|\mathbf{b}_k\|_2} \tag{15a}$$

$$\frac{\|\mathbf{x}_\lambda-\tilde{\mathbf{x}}_\lambda\|_2}{\|\mathbf{x}_\lambda\|_2} \leqq \frac{\sigma_1}{2\lambda}\frac{\|\mathbf{e}\|_2}{\|\mathbf{b}_\lambda\|_2} \tag{15b}$$

where $\mathbf{b}_k = A\mathbf{x}_k$ and $\mathbf{b}_\lambda = A\mathbf{x}_\lambda$.

Proof. It is elementary to derive (15a) from the inequalities $\|\mathbf{x}_k-\tilde{\mathbf{x}}_k\|_2 = \|V\Sigma_k^+U^T\mathbf{e}\|_2 \leqq \|\Sigma_k^+\|_2\|\mathbf{e}\|_2 = \sigma_k^{-1}\|\mathbf{e}\|_2$ and $\|\mathbf{b}_k\|_2 = \|A\mathbf{x}_k\|_2 \leqq \|A\|_2\|\mathbf{x}_k\|_2 = \sigma_1\|\mathbf{x}_k\|_2$. Further, it follows that $\|\mathbf{x}_\lambda-\tilde{\mathbf{x}}_\lambda\|_2 = \|V\Sigma_\lambda^+U^T\mathbf{e}\|_2 \leqq \|\Sigma_\lambda^+\|_2\|\mathbf{e}\|_2$, and assuming that $\sigma_n \leqq \lambda \leqq \sigma_1$ we obtain

$$\|\Sigma_\lambda^+\|_2 = \max_{1\leqq i\leqq n}\left\{\frac{\sigma_i}{\sigma_i^2+\lambda^2}\right\} \leqq \max_{\sigma_n\leqq\sigma\leqq\sigma_1}\frac{\sigma}{\sigma^2+\lambda^2} = 1/(2\lambda).$$

These two relations, together with $\|\mathbf{b}_\lambda\|_2 \leqq \sigma_1\|\mathbf{x}_\lambda\|_2$, lead directly to (15b). □

Remark. The bound in (15a) can also be derived from Lemma 2.3.2 of [8]. Applying the same lemma to Tikhonov regularization, we obtain a factor σ_1/λ instead of the factor $\sigma_1/(2\lambda)$ in (15b). If the error norms are bounded relative to $\|\mathbf{x}_o\|_2$ instead of

$\|\mathbf{x}_k\|_2$ and $\|\mathbf{x}_\lambda\|_2$, the right-hand sides simply change to $\sigma_1/\sigma_n\|\mathbf{e}\|_2/\|\mathbf{b}_o\|_2$ and $\sigma_1/(2\lambda)\|\mathbf{e}\|_2/\|\mathbf{b}_o\|_2$, where $\mathbf{b}_o = A\mathbf{x}_o$.

Theorem 4.1 confirms what has been observed experimentally and used in a number of applications, namely, that it is possible to choose k and λ such that the approximate perturbed solutions $\tilde{\mathbf{x}}_k$ and $\tilde{\mathbf{x}}_\lambda$ are fairly insensitive to the perturbations in $\mathbf{b}$. The theorem also shows that when $\lambda \approx \sigma_k$, as we assumed in the previous section, then both methods are approximately equally sensitive to the perturbations. Note that there is always a trade-off between Theorem 4.1 and Theorem 3.1 in the choice of k and λ: when k is small and λ is large then the perturbation bounds are small, whereas the TSVD and regularization errors may be large, and vice versa.

The results in Theorem 4.1 can also be used to derive expressions for the *condition numbers* associated with TSVD and Tikhonov regularization. Here, we shall use the following definitions.

DEFINITION 4.2. The condition numbers κ_k and κ_λ, associated with TSVD and Tikhonov regularization, respectively, are defined as

$$\kappa_k \equiv \lim_{\|e\|_2\to 0} \sup \frac{\|\mathbf{x}_k - \tilde{\mathbf{x}}_k\|_2}{\|\mathbf{x}_k\|_2}, \qquad \kappa_\lambda \equiv \lim_{\|e\|_2\to 0} \sup \frac{\|\mathbf{x}_\lambda - \tilde{\mathbf{x}}_\lambda\|_2}{\|\mathbf{x}_\lambda\|_2}, \tag{16}$$

where $\mathbf{x}_k$ and $\mathbf{x}_\lambda$ are the unperturbed TSVD and regularized solutions, and $\tilde{\mathbf{x}}_k$ and $\tilde{\mathbf{x}}_\lambda$ are the solutions when the right-hand side is perturbed by $\mathbf{e}$.
This definition, together with Theorem 4.1, immediately leads to:

COROLLARY 4.3. *The condition numbers* (16) *associated with* TSVD *and Tikhonov regularization are*

$$\kappa_k = \frac{\sigma_1}{\sigma_k}, \qquad \kappa_\lambda = \frac{\sigma_1}{2\lambda}. \tag{17}$$

Remark. Schock [18] recently derived another condition number $\bar{\kappa}_\lambda = \lambda/\sigma_n$ associated with Tikhonov regularization, based on the usual condition number of the matrix $A_\lambda^I = (A^TA + \lambda^2 I)^{-1}A^T$, which is the unique matrix that produces the regularized solution: $\mathbf{x}_\lambda = A_\lambda^I\mathbf{b}$. The result λ/σ_n is, however, not correct. Instead it should be

$$\bar{\kappa}_\lambda = \begin{cases} \sigma_1/\lambda, & \lambda \leqq \sqrt{\sigma_1\sigma_n} \\ \lambda/\sigma_n, & \lambda > \sqrt{\sigma_1\sigma_n} \end{cases} \tag{18}$$

which is easily derived from the proof of [18, Thm. 2]. We feel that our condition number κ_λ (18) is more correct than $\bar{\kappa}_\lambda$, since the matrix A_λ^I should not be used to compute $\mathbf{x}_\lambda$ numerically.

Next, we shall give the general perturbation bounds for $\mathbf{x}_k$ and $\mathbf{x}_\lambda$ when both the matrix A and the right-hand side $\mathbf{b}$ are perturbed. The perturbation of A may, e.g., arise from the approximations used to derive A from the kernel K in the numerical treatment of the integral equation (2).

THEOREM 4.4. *Let E and* $\mathbf{e}$ *denote the perturbations of A and* $\mathbf{b}$, *respectively, and assume that* $\|E\|_2 < \sigma_k - \sigma_{k+1}$ *and* $\|E\|_2 < \lambda$. *Then the perturbations of* $\mathbf{x}_k$ *and* $\mathbf{x}_\lambda$ *are bounded by*

$$\frac{\|\mathbf{x}_k - \tilde{\mathbf{x}}\|_2}{\|\mathbf{x}_k\|_2} \leqq \frac{\|A\|_2}{\sigma_k - \|E\|_2}\left[\left(2 + \frac{\sigma_{k+1}}{\omega_k}\right)\frac{\|E\|_2}{\|A\|_2} + \frac{\|\mathbf{e}\|_2}{\|\mathbf{b}_k\|_2} + \frac{\|E\|_2}{\omega_k}\frac{\|\mathbf{r}_k\|_2}{\|\mathbf{b}_k\|_2}\right] \tag{19a}$$

$$\frac{\|\mathbf{x}_\lambda - \tilde{\mathbf{x}}_\lambda\|_2}{\|\mathbf{x}_\lambda\|_2} < \frac{\|A\|_2}{\lambda - \|E\|_2}\left[2\frac{\|E\|_2}{\|A\|_2} + \frac{\|\mathbf{e}\|_2}{\|\mathbf{b}_\lambda\|_2} + \frac{\|E\|_2}{\lambda}\frac{\|\mathbf{r}_\lambda\|_2}{\|\mathbf{b}_\lambda\|_2}\right], \tag{19b}$$

where $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$, $\mathbf{r}_\lambda = \mathbf{b} - A\mathbf{x}_\lambda$, *and* $\omega_k = \sigma_k - \sigma_{k+1} - \|E\|_2$.

Proof. Equation (19a) follows immediately from [9, Thm. 3.4] together with the relation

$$\frac{\|E\|_2/\sigma_k}{1-\|E\|_2/\sigma_k-\sigma_{k+1}/\sigma_k}=\frac{\|A\|_2}{\sigma_k-\|E\|_2}\frac{\sigma_k-\|E\|_2}{\omega_k}\frac{\|E\|_2}{\|A\|_2}=\frac{\|A\|_2}{\sigma_k-\|E\|_2}\left(1+\frac{\sigma_{k+1}}{\omega_k}\right)\frac{\|E\|_2}{\|A\|_2}.$$

To prove (19b), we remind the reader that $\mathbf{x}_\lambda$ is the unique least squares solution to the problem min $\|C\mathbf{x}-\mathbf{d}\|_2$, where $C=\left[\begin{smallmatrix}A\\ \lambda I\end{smallmatrix}\right]$ and $\mathbf{d}=\left[\begin{smallmatrix}\mathbf{b}\\ \mathbf{0}\end{smallmatrix}\right]$. The matrix C has full rank for all $\lambda>0$, and we can apply the standard perturbation bound for least squares solutions [2, Thm. 5.5] to get

$$\|\mathbf{x}_k-\bar{\mathbf{x}}_\lambda\|_2\leqq\frac{\|A\|_2\|C^+\|_2}{1-\|E\|_2\|C^+\|_2}\left(\frac{\|E\|_2}{\|A\|_2}\|\mathbf{x}_\lambda\|_2+\frac{\|\mathbf{e}\|_2}{\|A\|_2}+\|E\|_2\|C^+\|_2\frac{\|\mathbf{d}-C\mathbf{x}_\lambda\|_2}{\|A\|_2}\right).$$

From the definitions of C and $\mathbf{d}$ we get $\|C^+\|_2=\sigma_n(C)^{-1}<\lambda^{-1}$, $\|A\|_2\|x_\lambda\|_2\geqq\|Ax_\lambda\|_2=\|\mathbf{b}_\lambda\|_2$ and $\|\mathbf{d}-C\mathbf{x}_\lambda\|_2=\left\|\begin{bmatrix}\mathbf{r}_\lambda\\ \lambda\mathbf{x}_\lambda\end{bmatrix}\right\|_2\leqq\|\mathbf{r}_\lambda\|_2+\lambda\|\mathbf{x}_\lambda\|_2$ such that

$$\frac{\|A\|_2\|C^+\|_2}{1-\|E\|_2\|C^+\|_2}<\frac{\|A\|_2}{\lambda-\|E\|_2},\quad\frac{\|\mathbf{e}\|_2}{\|A\|_2\|\mathbf{x}_\lambda\|_2}<\frac{\|\mathbf{e}\|_2}{\|\mathbf{b}_\lambda\|_2}$$

$$\|E\|_2\|C^+\|_2\frac{\|\mathbf{d}-C\mathbf{x}_\lambda\|_2}{\|A\|_2\|\mathbf{x}_\lambda\|_2}<\frac{\|E\|_2}{\lambda}\frac{\|\mathbf{r}_\lambda\|_2}{\|\mathbf{b}_\lambda\|_2}+\frac{\|E\|_2}{\|A\|_2}.$$

These relations then yield (19b). □

Although the bound in (19b) is not tight, it does illustrate our major point, namely, that the general perturbation bounds for $\mathbf{x}_k$ and $\mathbf{x}_\lambda$ are very similar whenever $\lambda\approx\sigma_k$ (which we have already assumed), provided that k and λ are chosen such that $\|E\|_2<\sigma_k-\sigma_{k+1}$ and $\|E\|_2<\lambda$. We see from (19a) that truncation of the sum in (8) at a nearly multiple singular value σ_k should be avoided; but apart from this, the results in Theorem 4.1 do not impose any particular requirement on the singular value spectrum. That is, we can actually truncate the expression (8) for x_k at any value of k, as long as σ_k is not nearly multiple.

The main conclusion to be drawn from Theorems 3.1, 3.2, 4.1, and 4.4 is therefore that the success of TSVD (as well as Tikhonov regularization) primarily depends on satisfaction of the DPC and, in fact, has little to do with the existence of a gap in the singular value spectrum of A. If, for some k, there is a large gap between σ_k and σ_{k+1} (i.e., A has well-determined numerical rank), then this k is usually identical to the numerical rank of A and it is therefore often convenient to truncate the expression for $\mathbf{x}_k$ at this k [9]. In this case, it is natural to require the DPC satisfied for the first k coefficients β_i only, and to consider the remaining β_i-coefficients associated with the residual. If, on the other hand, A has ill-determined numerical rank, then there is no point in trying to find A's numerical rank. Instead, one should choose k in order to suppress, as much as possible, the influence of the perturbations while, at the same time, keeping the TSVD error as small as possible. In the next section, we shall discuss this in more detail.

5. Characterization of the solutions. In this section we are mainly interested in the TSVD solution and the regularized solution when they are influenced by perturbations of the right-hand side. In order to understand the influence of such perturbations and to be able to select a proper truncation parameter k and regularization parameter λ, we therefore seek to characterize the behavior of the perturbed solutions $\bar{\mathbf{x}}_k$ and $\bar{\mathbf{x}}_\lambda$ as functions of k and λ. A convenient way to characterize any solution to the least squares problem (1) is to plot its norm versus the norm of the corresponding residual, as suggested in [13, Chap. 26]. Since we are only interested in that component of the residual which lies in the column space of A, we define the residuals $\mathbf{r}_k$ and $\mathbf{r}_\lambda$ corresponding to $\mathbf{x}_k$ and $\mathbf{x}_\lambda$ by

$$(20)\qquad \mathbf{r}_k \equiv \mathbf{b}_o - A\mathbf{x}_k = \sum_{i=k+1}^{n} \mathbf{u}_i^T\mathbf{b}\,\mathbf{u}_i, \qquad \mathbf{r}_\lambda \equiv \mathbf{b}_o - A\mathbf{x}_\lambda = \sum_{i=1}^{n} \frac{\lambda^2}{\sigma_i^2+\lambda^2}\mathbf{u}_i^T\mathbf{b}\,\mathbf{u}_i.$$

It can be proved that for regularization $\|\mathbf{x}_\lambda\|_2$ is a decreasing function of $\|\mathbf{r}_\lambda\|_2$, while for TSVD $\|\mathbf{x}_k\|_2$ is a decreasing function of $\|\mathbf{r}_k\|_2$ on a finite set [10, Thm. 5.3], and that the points $(\|\mathbf{r}_k\|_2, \|\mathbf{x}_k\|_2)$, $k=1,\cdots,n-1$ always lie above the curve $(\|\mathbf{r}_\lambda\|_2, \|\mathbf{x}_\lambda\|_2)$ [13, Thm. (25.49)]. The distance between these points and the curve was already analyzed in Theorem 3.2. Here, we shall give a more detailed (although not strictly rigorous) description of the curve and the set of points.

First, let us consider the behavior of the curve $(\|\mathbf{r}_\lambda\|_2, \|\mathbf{x}_\lambda\|_2)$, with $\mathbf{x}_\lambda$ and $\mathbf{r}_\lambda$ given by (7) and (20). Obviously, $\mathbf{x}_\lambda \to \mathbf{0}$ and $\mathbf{r}_\lambda \to \mathbf{b}_o$ as $\lambda \to \infty$, whereas $\mathbf{x}_\lambda \to \mathbf{x}_o$ and $\mathbf{r}_\lambda \to \mathbf{0}$ as $\lambda \to 0$. For $\lambda \ll \sigma_n$, we have

$$\sigma_i^2+\lambda^2 \approx \sigma_i^2 \Rightarrow \|\mathbf{r}_\lambda\|_2 \approx \lambda^2\left(\sum_{i=1}^{n}(\mathbf{u}_i^T\mathbf{b}/\sigma_i^2)^2\right)^{1/2} \leqq \left(\frac{\lambda}{\sigma_n}\right)^2\|\mathbf{b}_o\|_2$$

and $\sigma_i^2/(\sigma_i^2+\lambda^2)\approx 1 \Rightarrow \mathbf{x}_\lambda \approx \mathbf{x}_o$; i.e., for small λ the curve $(\|\mathbf{r}_\lambda\|_2, \|\mathbf{x}_\lambda\|_2)$ is approximately a horizontal line at $\|\mathbf{x}_\lambda\|_2 \approx \|\mathbf{x}_o\|_2$. When λ increases, we see that $\|\mathbf{x}_\lambda\|_2$ starts to decrease while $\|\mathbf{r}_\lambda\|_2$ still grows towards $\|\mathbf{b}_o\|_2$, and thus the curve must bend down towards the abscissa axis. The value of λ for which $\|\mathbf{x}_\lambda\|_2$ markedly starts to bend down depends on the σ_i as well as the $\mathbf{u}_i^T\mathbf{b}$. If the DPC is satisfied, such that the sum in the expression (7) for $\mathbf{x}_\lambda$ is dominated by its first terms, then obviously λ must be comparable with the largest singular values σ_i to significantly influence $\mathbf{x}_\lambda$. For such values of λ, $\|\mathbf{r}_\lambda\|_2$ will be somewhat smaller than $\|\mathbf{b}_o\|_2$ because the coefficients to the first terms in the expression (20) for $\mathbf{r}_\lambda$ will be less than one. If the DPC is not satisfied, we can assume that most of the terms in the expression for $\mathbf{x}_\lambda$ actually contribute to this vector, and the influence of λ can be felt for much smaller values of λ than before. Thus, the curve also starts to bend down for smaller values of $\|\mathbf{r}_\lambda\|_2$ than before.

Next, we consider the points $(\|\mathbf{r}_k\|_2, \|\mathbf{x}_k\|_2)$. If the DPC is satisfied, Theorem 3.2 guarantees that for large k there always exists a $\lambda \in [\sigma_{k+1}, \sigma_k]$ such that the points are close to the curve $(\|\mathbf{r}_\lambda\|_2, \|\mathbf{x}_\lambda\|_2)$, whereas for small k we cannot guarantee this. However, we can see from the expressions (8) and (20) for $\mathbf{x}_k$ and $\mathbf{r}_k$ that their norms must behave quantitatively like $\|\mathbf{x}_\lambda\|_2$ and $\|\mathbf{r}_\lambda\|_2$. That is, if the DPC is satisfied, then for large k the points will approximately be on the same horizontal line as the curve, and as k gets smaller the points will eventually start to bend down, always lying strictly above the curve. If the DPC is not satisfied, the points will also deviate from the curve for large k.

If we make more assumptions about the case when the right-hand side does not satisfy the DPC, we can also say more about the curve and the points. This is particularly relevant for the perturbation $\mathbf{e}$ of $\mathbf{b}$. An interesting case (see below) is when all the coefficients $|\mathbf{u}_i^T\mathbf{e}|$ of the perturbation are approximately of the same size,

$$|\mathbf{u}_i^T\mathbf{e}| \approx \varepsilon_0, \qquad i=1,\cdots,m.$$

The corresponding "solution" $\mathbf{x}_\lambda^{(e)}$ and "residual" $\mathbf{r}_\lambda^{(e)}$ are given by

$$(21)\qquad \mathbf{x}_\lambda^{(e)} \approx \varepsilon_o \sum_{i=1}^{n} \frac{\sigma_i}{\sigma_i^2+\lambda^2}\mathbf{v}_i, \qquad \mathbf{r}_\lambda^{(e)} \approx \varepsilon_o \sum_{i=1}^{n} \frac{\lambda^2}{\sigma_i^2+\lambda^2}\mathbf{u}_i.$$

The vector $\mathbf{x}_\lambda^{(e)}$ is called the "noise amplification error" in [1]. In particular, the norm of $\mathbf{x}_o^{(e)} = A^+\mathbf{e}$ satisfies $\varepsilon_o/\sigma_n \lesssim \|\mathbf{x}_o^{(e)}\|_2 \lesssim \sqrt{n}\,\varepsilon_o/\sigma_n$. We see from (21) that for λ in the range $\sigma_n \leqq \lambda < \infty$, $\|\mathbf{r}_\lambda^{(e)}\|_2$ varies in the quite small range from approximately ε_o to $\sqrt{n}\,\varepsilon_o$. Concerning $\mathbf{x}_\lambda^{(e)}$, the sum in (21) is dominated by just a few, say p, of the terms, namely, those for which $\sigma_i \approx \lambda$ and $\sigma_i/(\sigma_i^2+\lambda^2) \approx 1/(2\lambda)$ such that $\|\mathbf{x}_\lambda^{(e)}\|_2 \approx p \cdot \varepsilon_o/(2\lambda) \approx \varepsilon_o/\lambda$. Thus, as $\lambda \to \infty$, the curve $(\|\mathbf{r}_\lambda^{(e)}\|_2, \|\mathbf{x}_\lambda^{(e)}\|_2)$ soon becomes almost a vertical line at $\|\mathbf{r}_\lambda^{(e)}\|_2 \approx \sqrt{n}\,\varepsilon_o$. Exactly the same conclusions hold for the points $(\|\mathbf{r}_k^{(e)}\|_2, \|\mathbf{x}_k^{(e)}\|_2)$ obtained when applying TSVD to $\mathbf{e}$.

Typical examples of both curves $(\|\mathbf{r}_\lambda\|_2, \|\mathbf{x}_\lambda\|_2)$, with the DPC satisfied, and $(\|\mathbf{r}_\lambda^{(e)}\|_2, \|\mathbf{x}_\lambda^{(e)}\|_2)$, with the DPC not satisfied, are shown in Fig. 1 for the situation $\|\mathbf{e}\|_2 < \|\mathbf{b}_o\|_2$. Note that the latter curve starts to bend down towards the abscissa axis before the first curve does (e.g., for smaller λ). Also note that the level $\|\tilde{\mathbf{x}}_o\|_2 \approx \|\mathbf{x}_o^{(e)}\|_2 \approx \sqrt{n}\,\varepsilon_o/\sigma_n$ lies over the level $\|\mathbf{x}_o\|_2$.

We are now ready to describe the behavior of the solutions $\tilde{\mathbf{x}}_\lambda = \mathbf{x}_\lambda + \mathbf{x}_\lambda^{(e)}$ and $\tilde{\mathbf{x}}_k = \mathbf{x}_k + \mathbf{x}_k^{(e)}$ under the influence of errors $\mathbf{e}$ in the right-hand side. We make the following assumptions in order to be able to carry out a meaningful analysis.

ASSUMPTION 5.1. *Let* $\mathbf{e}$ *be a perturbation of the right-hand side* $\mathbf{b}$ *in* (1). *We assume the following*:

1. *The unperturbed right-hand side* $\mathbf{b}$ *satisfies the* DPC.
2. *The perturbation* $\mathbf{e}$ *is a random vector of zero mean and covariance matrix* $\varepsilon_o^2 I$.
3. *The norm of* $\mathbf{e}$ *satisfies* $\|\mathbf{e}\|_2 < \|\mathbf{b}_o\|_2$.

The first assumption is necessary for the convergence of the methods. The second assumption is very common in least squares problems. If it is not satisfied, we should either scale the equations (if the covariance matrix is diagonal) or use the general Gauss-Markoff linear model for a general covariance matrix (cf., e.g., [2, § 14] and [26]). As a consequence of this assumption, the expected value of all $\|\mathbf{u}_i^T\mathbf{e}\|_2 = \sqrt{n}\,\varepsilon_o$ independently of i. Assumption 3 is simply a requirement that the signal-to-noise ratio

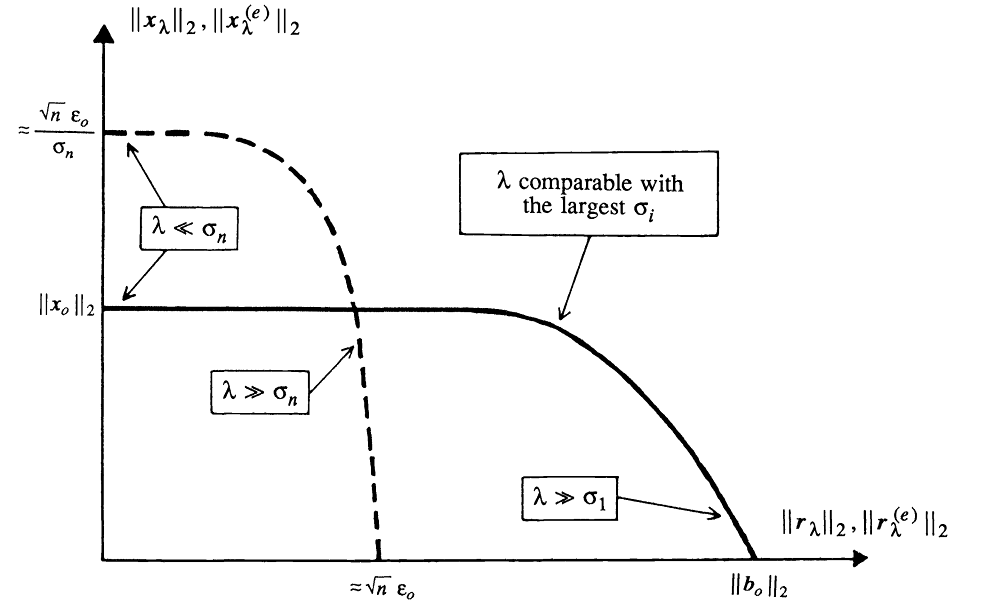


FIG. 1. *Typical behavior of* —— *the regularized solution* $\mathbf{x}_\lambda$ *for a right-hand side* $\mathbf{b}$ *that satisfies the discrete Picard condition, and* - - - *the regularized solution* $\tilde{\mathbf{x}}_\lambda^{(e)}$ *for a right-hand side* $\mathbf{e}$ *that does not.*

in the given right-hand side is not too large to let one retrieve a satisfactory approximate solution.

With these assumptions it then follows from the above analysis that the curve $(\|\tilde{\mathbf{r}}_\lambda\|_2, \|\tilde{\mathbf{x}}_\lambda\|_2)$ and the set of points $(\|\tilde{\mathbf{r}}_k\|_2, \|\tilde{\mathbf{x}}_k\|_2)$ will appear as shown in Fig. 2. For $\lambda \ll \sigma_n$, the curve is almost horizontal at the level $\|\tilde{\mathbf{x}}_\lambda\|_2 \approx \|\mathbf{x}_o^{(e)}\|_2 \approx \sqrt{n}\,\varepsilon_o/\sigma_n$, since the regularized solution $\tilde{\mathbf{x}}_\lambda$ is dominated by the term $\mathbf{x}_\lambda^{(e)}$. Increasing λ, the curve soon starts to bend down due to the influence of λ in the term $\mathbf{x}_\lambda^{(e)}$, which still dominates $\tilde{\mathbf{x}}_\lambda$. This part of the curve therefore resembles the dashed line in Fig. 1. Increasing λ further, the other term $\mathbf{x}_\lambda$ will start to dominate $\tilde{\mathbf{x}}_\lambda$ at some point such that the curve now stays at a new level at $\|\tilde{\mathbf{x}}_\lambda\|_2 \approx \|\mathbf{x}_o\|_2$, until it eventually starts to bend down again for λ comparable with the largest singular values. This part of the curve is therefore similar to the solid line in Fig. 1. As long as Assumption 5.1 is satisfied, there will always be a more or less distinct "corner" somewhere in the middle of the curve, where the dominating term in $\tilde{\mathbf{x}}_\lambda$ switches from $\mathbf{x}_\lambda$ to $\mathbf{x}_\lambda^{(e)}$. The set of points corresponding to the TSVD solution $\tilde{\mathbf{x}}_k$ behaves in exactly the same way, also exhibiting a "corner" when the dominating term in $\tilde{\mathbf{x}}_k$ switches from $\mathbf{x}_k$ to $\mathbf{x}_k^{(e)}$. At this corner, and to its right, the component $\mathbf{x}_k$ dominates and Theorem 3.2 ensures that the points will be close to the solid curve (except for the smallest k). To the left, $\mathbf{x}_k^{(e)}$ dominates, and as k increases the points will deviate from the solid curve. We can summarize the main result as follows.

CHARACTERIZATION 5.2. *If Assumption* 5.1 *is satisfied, then the curve* $(\|\tilde{\mathbf{r}}_\lambda\|_2, \|\tilde{\mathbf{x}}_\lambda\|_2)$ *as well as the set of points* $(\|\tilde{\mathbf{r}}_k\|_2, \|\tilde{\mathbf{x}}_k\|_2)$ *exhibit a "corner" behavior as functions of their parameters* λ *and* k. *Both "corners" occur approximately at* $(\sqrt{n}\,\varepsilon_o, \|\mathbf{x}_o\|_2)$. *The larger the difference between the decay rates of* $|\mathbf{u}_i^T\mathbf{b}|$ *and* $|\mathbf{u}_i^T\mathbf{e}|$, *the more distinct the "corners" will appear.*

The plots in Fig. 2 provide a natural choice of the regularization parameter λ and the truncation parameter k that must be selected. It is obvious that the optimal values of λ and k are those that yield solutions near the "corners" of the curve and the point set, respectively, since these solutions approximate $\mathbf{x}_o$ as closely as possible without being dominated by the contributions from the perturbation $\mathbf{e}$. In the case of the TSVD

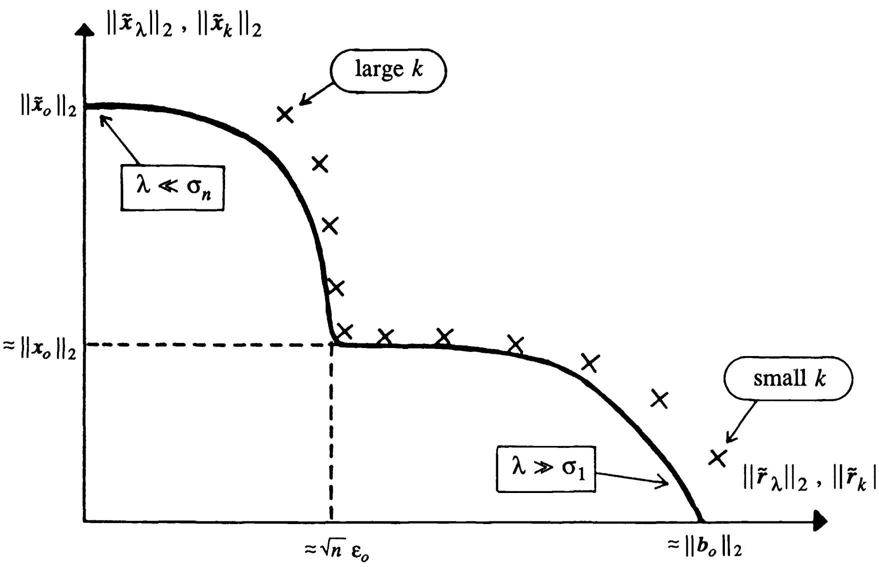


FIG. 2. *Comparison of* × TSVD *solutions* $\tilde{\mathbf{x}}_k$ *and* —— *regularized solutions* $\tilde{\mathbf{x}}_\lambda$ *corresponding to a perturbed right-hand side* $\mathbf{b}+\mathbf{e}$.

solution $\tilde{\mathbf{x}}_k$, we can also say that k should be chosen as large as possible, but with the constraint that the k coefficients $\mathbf{u}_i^T(\mathbf{b}+\mathbf{e})$, $i=1,\cdots,k$ in the truncated sum for $\tilde{\mathbf{x}}_k$ satisfy the DPC. The optimal solutions, produced by these optimal values of λ and k, satisfy $\|\tilde{\mathbf{x}}_\lambda\|_2 \approx \|\tilde{\mathbf{x}}_k\|_2 \approx \|\mathbf{x}_o\|_2$ and $\|\tilde{\mathbf{r}}_\lambda\|_2 \approx \|\tilde{\mathbf{r}}_k\|_2 \approx \|\mathbf{e}\|_2$; i.e., they are *reasonable solutions* in the terminology of Varah [24]. We stress that whenever Assumption 5.1 is satisfied, and λ and k are chosen as described above, then both TSVD and Tikhonov regularization are guaranteed to produce very similar reasonable solutions that *converge* to $\mathbf{x}_o$ as $\mathbf{e}\to\mathbf{0}$.

We shall not go into a detailed description of numerical methods for determining λ and k according to the above criteria. Suffice it to say that the key idea in most of these methods is actually to locate the "corners" of the curve and the point set as illustrated in Fig. 2. This is clearly the case in methods based on the *discrepancy principle* [8, § 3.3] where we increase k or decrease λ until the residual norm (or some function of this norm) is of the same size as the norm of the errors $\|\mathbf{e}\|_2 \approx \sqrt{n}\varepsilon_o$. *Generalized cross-validation* [7] is a promising alternative method which, in the case of TSVD, simply amounts to choosing k so as to minimize the function $\|\tilde{\mathbf{r}}_k\|_2^2/(m-n)^2$. This k is identical to the k for which $\|\tilde{\mathbf{r}}_k\|_2^2/(m-n)$, as a function of k, starts to level off and become an estimate of the variance $\|\mathbf{e}\|_2^2/m$ of the noise, and the corresponding point $(\|\tilde{\mathbf{r}}_k\|_2, \|\tilde{\mathbf{x}}_k\|_2)$ is therefore near the "corner" of the point set. The same holds for Tikhonov regularization (cf. the discussion in [11]).

6. Numerical examples. This section includes two examples of the numerical solution of first kind Fredholm integral equations (2), illustrating the discussion in the previous sections. In both examples we choose $m=n$, and we discretize by means of the method of moments with simple orthonormal basis functions ϕ_i chosen to give a piecewise constant approximation to f:

$$\phi_i(x)=\begin{cases} h^{-1/2}, & a+(i-1)h \leqq x \leqq a+ih \\ 0, & \text{otherwise} \end{cases} \qquad i=1,\cdots,n$$

where $h=(b-a)/n$ and $[a,b]$ is the integration interval. The elements of A and $\mathbf{b}$ in the least squares problem (1) are then given by the integrals

$$(22) \quad a_{ij}=h^{-1}\int_{a+(i-1)h}^{a+ih}\int_{a+(j-1)h}^{a+jh} K(s,x)\,dx\,ds, \qquad b_i=h^{-1/2}\int_{a+(i-1)h}^{a+ih} g(s)\,ds.$$

For more details on this method see, e.g., [10] where it is shown that the singular values σ_i of A, when computed by (22), are $O(n^{-2})$-approximations to the μ_i in the SVE of K.

The first example is a classical example by Phillips [16]. The integral equation is given by the following K and g:

$$K(s,x)=\begin{cases} 1+\cos[\pi(s-x)/3], & |s-x|\leqq 3 \\ 0, & |s-x|>3 \end{cases}$$

$$g(s)=(6-|s|)\left(1+\frac{1}{2}\cos\left(\frac{\pi s}{3}\right)\right)+\frac{9}{2\pi}\sin\left(\frac{\pi|s|}{3}\right)$$

with $[a,b]=[-6,6]$. The solution is $f(x)=1+\cos(\pi x/3)$. The explicit formulas for A and $\mathbf{b}$, evaluated by means of (22), are given in [10]. We used $n=64$ and perturbed the right-hand side $\mathbf{b}$ by random numbers from a normal distribution with zero mean and standard deviation 10^{-4} such that $\|\mathbf{e}\|_2 \approx 8\cdot 10^{-4}$. The condition number of A is $\sigma_1/\sigma_n = 2.8\cdot 10^5$, and we know that Assumption 5.1 is satisfied. For $i \leqq 13$ the computed quantities σ_i, $|\mathbf{u}_i^T\mathbf{b}|$ and $|\mathbf{u}_i^T\mathbf{b}|/\sigma_i$ (not shown here) all decay as expected. For $i \geqq 13$ the singular values σ_i continue to decrease while the coefficients $|\mathbf{u}_i^T\mathbf{b}|$ settle at the error level about 10^{-4}. Hence, the $|\mathbf{u}_i^T\mathbf{b}|/\sigma_i$ increase almost monotonically with i for $i \geqq 13$. Fig. 3 shows the curve $(\|\tilde{\mathbf{r}}_\lambda\|_2, \|\tilde{\mathbf{x}}_\lambda\|_2)$ and some of the points $(\|\tilde{\mathbf{r}}_k\|_2, \|\tilde{\mathbf{x}}_k\|_2)$; note the likeness with Fig. 2. It is clear, both from these plots and from plots of $|\mathbf{u}_i^T\mathbf{b}|/\sigma_i$, that we should truncate the TSVD solution $\mathbf{x}_k$ at about $k = 12$. The approximate solution, computed from $\tilde{\mathbf{x}}_k$, agrees with the true f within a maximum deviation of about 10^{-2}, which is largely due to the TSVD error $\mathbf{x}_o - \mathbf{x}_k$.

The second example is a real problem from observational astronomy, where the right-hand side g is the probability density function of *observed* stellar parallaxes, whereas f is the true probability density function of these parallaxes. Assuming that the measurement errors are normally distributed, it is easy to show that f is related to g by a first kind Fredholm integral equation (2) with

$$K(s,x) = \frac{1}{\rho\sqrt{2\pi}} \exp\left(-\frac{1}{2}\frac{(s-x)^2}{\rho^2}\right).$$

The factor ρ reflects the accuracy of the measurements. As a case study we used a standard set of observations from [19, Table 4, p. 30] defining g in the form of a piecewise constant function and with $\rho = 0.014234$. We used the interval $[a, b] = [-0.03, 0.10]$, and the elements of A (22) were calculated using the two-dimensional 9-point Simpson quadrature rule. The computed values of σ_i and $|\mathbf{u}_i^T\mathbf{b}|/\sigma_i$ are shown

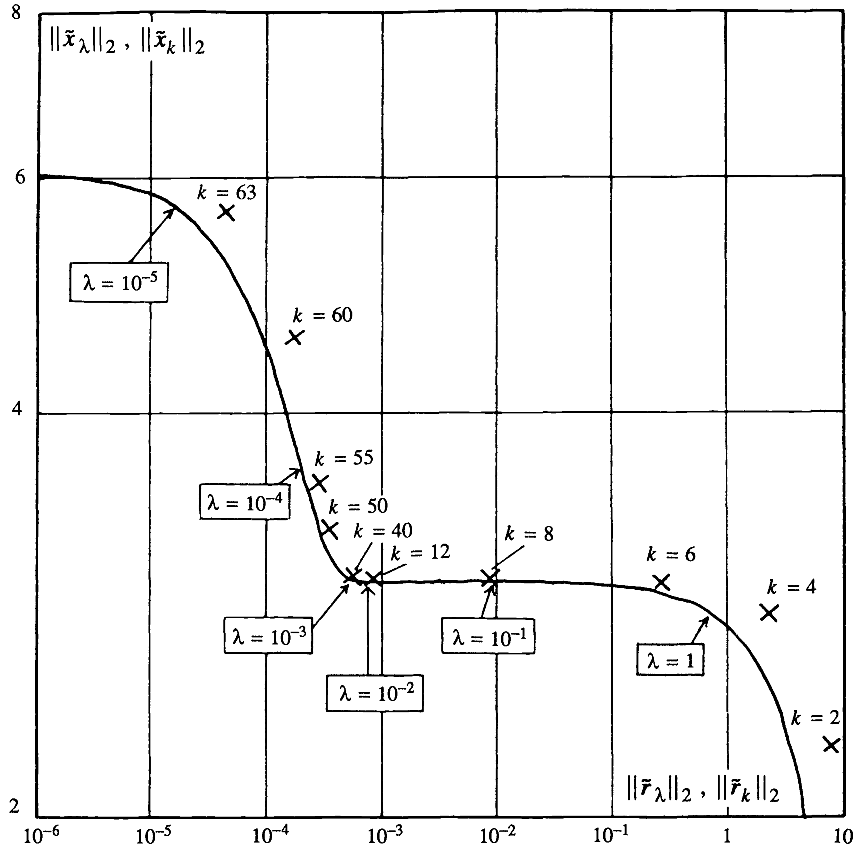


FIG. 3. *Plot of* TSVD *and regularized solutions for Phillips's example.*

in Fig. 4. Note that the $|\mathbf{u}_i^T\mathbf{b}|/\sigma_i$ initially decay slightly with i, and that they soon start to increase dramatically. It is evident from this figure that the TSVD should be truncated at k equal to 6 or 7. Fig. 5 shows plots of $(\|\tilde{\mathbf{r}}_\lambda\|_2, \|\tilde{\mathbf{x}}_\lambda\|_2)$ and $(\|\tilde{\mathbf{r}}_k\|_2, \|\tilde{\mathbf{x}}_k\|_2)$, and the similarity with the idealized plot in Fig. 2 indicates that Assumption 5.1 is actually satisfied. We see from Fig. 5 that the error level is about $\|\mathbf{e}\|_2 \approx 10^{-3}$, that there is a distinct "corner" on the curve, and that the TSVD solutions $\mathbf{x}_k$ with $k = 5, 6, 7, 8$ are

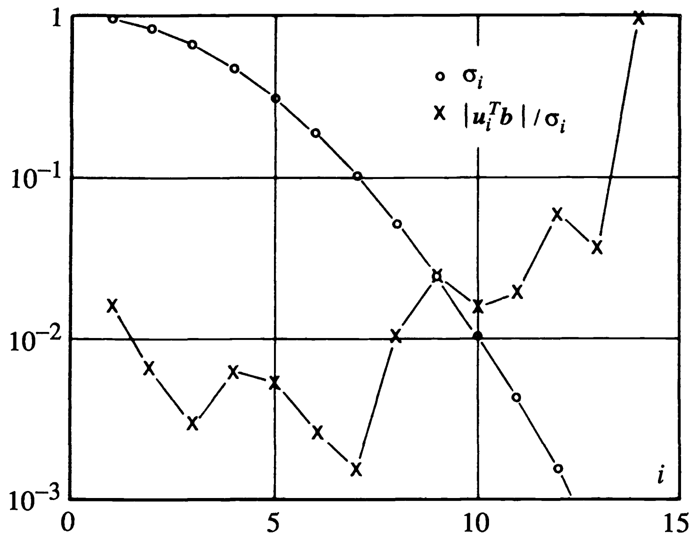


FIG. 4. *The computed singular values σ_i and coefficients $|\mathbf{u}_i^T\mathbf{b}|/\sigma_i$ for the second example with observed stellar parallaxes.*

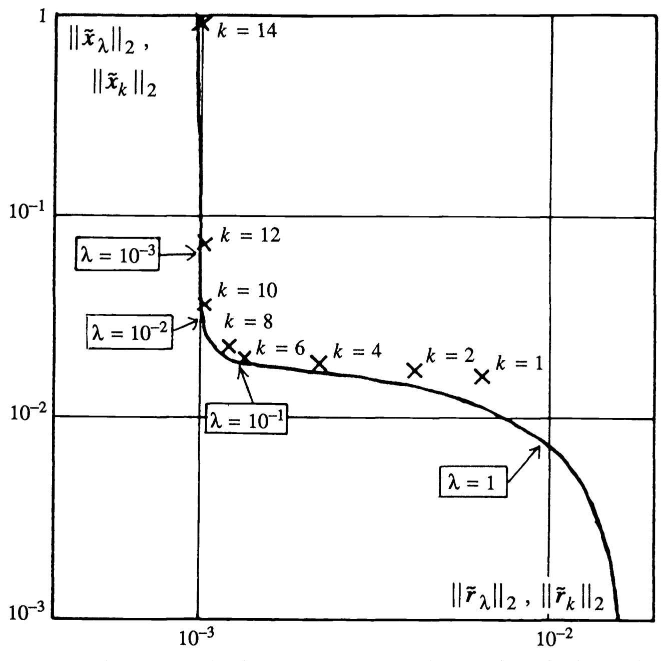


FIG. 5. *Plot of* TSVD *and regularized solutions for the second example with observed stellar parallaxes.*

close to this "corner." The computed solution corresponding to $k=6$ turned out to give the best results.

Acknowledgment. I would like to thank Professor Leslie Foster for discussions that led my investigations in the direction presented here.

REFERENCES

[1] C. M. AULICK AND T. M. GALLIE, *Isolating error effects in solving ill-posed problems*, SIAM J. Algebraic Discrete Methods, 4 (1983), pp. 371-376.

[2] Å. BJÖRCK, *Least squares methods*, in Handbook of Numerical Analysis, Vol. III: Finite Difference Methods—Solution of Equations in $\mathbb{R}^n$, P. G. Ciarlet and J. L. Lions, eds., Elsevier, Amsterdam, the Netherlands, 1989.

[3] I. J. D. CRAIG AND J. C. BROWN, *Inverse Problems in Astronomy*, Adam Hilger, Bristol, UK, 1986.

[4] J. J. M. CUPPEN, *Calculating the isochrones of ventricular depolarization*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 105-120.

[5] F. R. DE HOOG, *Review of Fredholm equations of the first kind*, in The Application and Numerical Solution of Integral Equations, R. S. Anderssen, F. R. de Hoog, and M. A. Lukas, eds., Sijthoff & Noordhoff, 1980, pp. 119-134.

[6] U. ECKHARDT AND K. MIKA, *Numerical treatment of incorrectly posed problems—A case study*, in Numerical Treatment of Integral Equations, J. Albrecht and L. Collatz, eds., Workshop on Numerical Treatment of Integral Equations, Oberwolfach, November 18-24, 1979, Birkhäuser Verlag, Boston, 1980, pp. 92-101.

[7] G. H. GOLUB, M. T. HEATH, AND G. WAHBA, *Generalized cross-validation as a method for choosing a good ridge parameter*, Technometrics, 21 (1979), pp. 215-223.

[8] C. W. GROETSCH, *The Theory of Tikhonov Regularization for Fredholm Integral Equations of the First Kind*, Pitman, Boston, 1984.

[9] P. C. HANSEN, *The truncated* SVD *as a method for regularization*, BIT, 27 (1987), pp. 534-553.

[10] ———, *Computation of the singular value expansion*, Computing, 40 (1988), pp. 185-199.

[11] ———, *Solution of ill-posed problems by means of truncated* SVD, in Numerical Mathematics, Singapore 1988, R. P. Agarwal, Y. M. Chow, and S. J. Wilson, eds., ISNM 86, Birkhäuser, Boston, (1988), pp. 179-192.

[12] ———, *Regularization,* GSVD *and truncated* GSVD, BIT, 29 (1989), pp. 491-504.

[13] C. L. LAWSON AND R. J. HANSON, *Solving Least Squares Problems*, Prentice Hall, Englewood Cliffs, NJ, 1974.

[14] P. LINTZ, *Uncertainty in the solution of linear operator equations*, BIT, 24 (1984), pp. 92-101.

[15] A. K. LOUIS AND F. NATTERER, *Mathematical problems of computerized tomography*, Proc. IEEE, 71 (1983), pp. 379-389.

[16] D. L. PHILLIPS, *A technique for the numerical solution of certain integral equations of the first kind*, J. Assoc. Comput. Mach., 9 (1962), pp. 84-97.

[17] G. R. RICHTER, *Numerical solution of integral equations of the first kind with nonsmooth kernels*, SIAM J. Numer. Anal., 15 (1978), pp. 511-522.

[18] E. SCHOCK, *What are the proper condition numbers of discretized ill-posed problems?* Linear Algebra Appl., 81 (1986), pp. 129-136.

[19] W. M. SMART, *Stellar Dynamics*, Cambridge University Press, London, UK, 1938.

[20] A. N. TIKHONOV, *Solution of incorrectly formulated problems and the regularization method*, Dok. Akad. Nauk SSSR, 151 (1963), pp. 501-504. (In Russian.) Soviet Math. Dokl. 4 (1963), pp. 1035-1038. (In English.)

[21] A. N. TIKHONOV AND A. V. GONCHARSKY, EDS., *Ill-Posed Problems in the Natural Sciences*, MIR Publishers, Moscow, 1987.

[22] J. M. VARAH, *On the numerical solution of ill-conditioned linear systems with applications to ill-posed problems*, SIAM J. Numer. Anal., 10 (1973), pp. 257-267.

[23] ———, *A practical examination of some numerical methods for linear discrete ill-posed problems*, SIAM Rev. 21 (1979), pp. 100-111.

[24] ———, *Pitfalls in the numerical solution of linear ill-posed problems*, SIAM J. Sci. Statist. Comput, 4 (1983), pp. 164-176.

[25] G. M. WING, *Condition numbers of matrices arising from the numerical solution of linear integral equations of the first kind*, J. Integral Equations, 9 (Suppl.) (1985), pp. 191-204.

[26] H. ZHA AND P. C. HANSEN, *Regularization and the general Gauss-Markoff linear model*, Math. Comp., to appear.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 3, pp. 519-530, May 1990

COMPUTING TRUNCATED SINGULAR VALUE DECOMPOSITION LEAST SQUARES SOLUTIONS BY RANK REVEALING QR-FACTORIZATIONS*

TONY F. CHAN† AND PER CHRISTIAN HANSEN‡

Abstract. Solutions to rank deficient least squares problems are conveniently expressed in terms of the singular value decomposition (SVD) of the coefficient matrix. When the matrix is *nearly* rank deficient, a common procedure is to neglect its smallest singular values, which leads to the truncated SVD (TSVD) solution. In this paper, an efficient method is presented for computing the TSVD solution via a QR-factorization, without the need for computing a complete SVD. The numerical rank of the matrix is determined by means of a rank revealing QR-factorization, which provides upper and lower bounds on the small singular values and approximations to the corresponding singular vectors, which are then refined by inverse subspace iteration and used in conjunction with the QR factors to compute the TSVD solution.

Key words. least squares problems, truncated singular value decomposition, nearly rank deficiency, numerical rank, rank revealing QR-factorization, inverse subspace iteration, subset selection

AMS(MOS) subject classifications. 65F25, 65F20

1. Introduction. In this paper, we consider an efficient and reliable numerical method for solving the linear least squares problem

$$\min \|A\mathbf{x}-\mathbf{b}\|_2, \qquad A \in \mathbb{R}^{m\times n}, \qquad m \geqq n \tag{1.1}$$

with the matrix A ill-conditioned and possibly rank deficient. This method is the *truncated* SVD (TSVD) *method*, which is based on the singular value decomposition (SVD) [10, § 2.3] of A:

$$A = U\Sigma V^T = \sum_{i=1}^{n} \mathbf{u}_i \sigma_i \mathbf{v}_i^T, \tag{1.2a}$$

where the left and right singular vectors $\mathbf{u}_i$ and $\mathbf{v}_i$ are the columns of the matrices U and V, respectively, and σ_i are the singular values of A. They are nonnegative and appear in nonincreasing order,

$$\sigma_1 \geqq \sigma_2 \geqq \cdots \geqq \sigma_n \geqq 0. \tag{1.2b}$$

The number of strictly positive singular values is the rank of A.

It is well known that the minimum norm least squares solution to (1.1) can be written in terms of the SVD of A as:

$$\mathbf{x} = \sum_{i=1}^{\operatorname{rank}(A)} \frac{\mathbf{u}_i^T\mathbf{b}}{\sigma_i}\mathbf{v}_i. \tag{1.3}$$

The ill-posed nature of (1.1), associated with the ill-conditioned matrix A, is caused by the appearance of one or more small singular values in the denominators of the sum in (1.3). Therefore, the basic idea of the TSVD method is to truncate the sums in

* Received by the editors December 15, 1986; accepted for publication (in revised form) February 1, 1989. This work was supported by the Department of Energy contract DE AC02 81ER 10996.

† Department of Mathematics, University of California, Los Angeles, California 90024.

‡ Danish Computing Center for Research and Education (UNI • C), Building 305, Technical University of Denmark, DK-2800 Lyngby, Denmark. The research of this author was sponsored by National Science Foundation grant DCR 8412314 and by a Fulbright supplementary grant.

(1.2a) and (1.3) at a value $k < \operatorname{rank}(A)$ such that all the small singular values are discarded. This corresponds to defining a new matrix A_k by

$$A_k \equiv \sum_{i=1}^{k} \mathbf{u}_i \sigma_i \mathbf{v}_i^T, \qquad k < \operatorname{rank}(A) \tag{1.4}$$

and substituting this matrix for A in (1.1), leading to a new least squares problem:

$$\min \|A_k \mathbf{x} - \mathbf{b}\|_2. \tag{1.5}$$

The minimum norm solution [10, §6.1] to this new problem is termed the TSVD *solution* $\mathbf{x}_k$ to the problem (1.1), and it is obviously obtained from (1.3) by truncation of the sum at k:

$$\mathbf{x}_k \equiv \sum_{i=1}^{k} \frac{\mathbf{u}_i^T \mathbf{b}}{\sigma_i} \mathbf{v}_i = \left(\sum_{i=1}^{k} \mathbf{v}_i \sigma_i^{-1} \mathbf{u}_i^T \right) \mathbf{b} = A_k^+ \mathbf{b}. \tag{1.6}$$

We refer to [11] and [12] for a discussion of the properties of the TSVD solution.

The matrix A_k^+ defined in (1.6) is the pseudoinverse of the matrix A_k in (1.4) and (1.5) (while it is an outer inverse of A). Therefore the problem of computing $\mathbf{x}_k$ is related, at least in theory, to the problem of computing the matrix A_k^+. Although we need not compute A_k^+ explicitly, (1.6) does emphasize the two quantities required to compute the TSVD solution, namely, the number $n-k$ of small singular values $\sigma_{k+1}, \cdots, \sigma_n$ to be discarded and the subspace spanned by the corresponding right singular vectors $\mathbf{v}_{k+1}, \cdots, \mathbf{v}_n$, to which the TSVD solution must be orthogonal. These two quantities also distinguish the TSVD solution from the standard QR-factorization-based procedure for rank deficient problems, such as HFTI by Lawson and Hanson [15], which relies on the heuristic procedure of column pivoting for identifying the numerical rank from the appearance of small elements in the bottom part of the computed R matrix. In § 2, we return to the problem of choosing the truncation parameter and describe a method, based on the singular value spectrum of A, for selecting a proper value of k.

While we can compute $\mathbf{x}_k$ from the SVD via (1.6), this is an expensive procedure. Computation of Σ and V in the SVD (1.2a) of A involves about $(m + 17/3n)n^2$ flops [10, Table 6.5-1]. In this paper we demonstrate how to compute the TSVD solution from *any* QR-*factorization* of A with much less computational effort: approximately $(m+p)n^2$ flops, where p is a small integer. Column pivoting during the QR-factorization of A is not required, thus making our algorithm particularly suited for sparse matrices. An outline of our algorithm is as follows. From any QR-factorization of A, we compute a rank-revealing QR-factorization (RRQR) [2], from which upper and lower bounds for the small singular values are easily obtained. This is used to compute the numerical rank k of A. The RRQR also yields approximations to the last $n-k$ singular vectors, which are then improved by means of inverse subspace iteration. Finally, these vectors and the above RRQR are both used to compute the TSVD solution.

It may seem surprising, compared to the work involved in a complete SVD computation, that we can compute $\mathbf{x}_k$ much more "cheaply." It is possible to avoid most of the computational effort in computing the complete SVD of A because only a part of the information, provided by the complete SVD, is actually required to compute the TSVD solution: namely, the $n-k$ smallest singular values and the corresponding singular vectors. And this information can, as we shall see, be extracted from a QR-factorization of A.

The TSVD solution is related to some other solutions to (1.1) obtained by other methods. For example, it is similar to the so-called "deflated solutions" as defined by

Chan [1], as well as the solution obtained by "regularization in standard form" as shown by Hansen [11], [12]. It is, however, generally different from the so-called basic solution (cf., e.g., [10, § 6.4]) computed by the above-mentioned HFTI procedure and also different from the solution obtained by "subset selection" as described by Golub, Klema, and Stewart [9], although the present algorithm may also be used as a first step in performing subset selection (see § 6). We stress that the purpose of this paper is to present an efficient method for computing the TSVD solution, but not to emphasize the comparison of this solution to other solutions.

2. The numerical rank and the general least squares solution. In this section we consider the selection of a proper value of the truncation parameter k in the truncated SVD expansion (1.3). The usual approach is to let k be equal to the *numerical rank* of A, defined as the number of singular values of A strictly larger than a certain threshold τ:

$$\sigma_1 \geqq \cdots \geqq \sigma_k > \tau \geqq \sigma_{k+1}. \tag{2.1}$$

This is a reasonable approach as long as there is a well-defined gap between the singular values σ_k and σ_{k+1}, since in this case the numerical rank k is well determined with respect to τ. If, however, $\sigma_k \approx \sigma_{k+1}$, then the numerical rank is not well determined with respect to τ. This problem is also reflected in the perturbation theory for the TSVD. Let E be a perturbation of A in (1.1) such that $\|E\|_2 < \sigma_k - \sigma_{k+1}$. Then it is shown in [11] that the perturbed pseudoinverse $(A+E)_k^+$ satisfies

$$\frac{\|A_k^+ - (A+E)_k^+\|_2}{\|A_k^+\|_2} \leqq 3 \frac{\eta_k}{(1-\eta_k)(1-\eta_k-\omega_k)}, \tag{2.2a}$$

where we have defined

$$\eta_k \equiv \frac{\|E\|_2}{\sigma_k}, \qquad \omega_k \equiv \frac{\sigma_{k+1}}{\sigma_k}. \tag{2.2b}$$

From this result it is seen that a small upper bound on the perturbation of the pseudoinverse A_k^+ (and thus on the TSVD solution $\mathbf{x}_k$) depends not only on a small η_k, but also on a small ω_k, which means that there must be a distinct gap in the singular value spectrum between σ_k and σ_{k+1}. As we shall see in § 4, this requirement on ω_k also enters into our algorithm. We will therefore make the assumption that the numerical rank k is well defined.

Since we have discarded $n-k$ singular values of A, such that the matrix A_k (1.4) has exact rank k, there are exactly $n-k$ linearly independent solutions to the homogeneous problem associated with (1.5), and these are linear combinations of the last $n-k$ right singular vectors of A. Therefore, the *general solution* to (1.5) can formally be written as

$$\mathbf{x}_{GS} = \mathbf{x}_k + \sum_{i=k+1}^{n} c_i \mathbf{v}_i, \qquad c_i \text{ arbitrary}, \tag{2.3}$$

in which $\mathbf{v}_{k+1}, \cdots, \mathbf{v}_n$ are the last $n-k$ right singular vectors of the matrix V in (1.2). The TSVD solution $\mathbf{x}_k$ (1.6) is the unique solution to (1.5), among all $\mathbf{x}_{GS}$, which minimizes $\|\mathbf{x}_{GS}\|_2$ (hence the name "minimum norm solution"). Define the matrix $V_o = [\mathbf{v}_{k+1} \cdots \mathbf{v}_n] \in \mathbb{R}^{n\times(n-k)}$. From (2.1) it immediately follows that

$$\|AV_0\|_2 < \tau. \tag{2.4}$$

Hence, the subspace spanned by these vectors will be termed the *numerical null-space* N_k *of* A:

$$N_k \equiv \operatorname{span}\{\mathbf{v}_{k+1}, \cdots, \mathbf{v}_n\}. \tag{2.5}$$

The general solution $\mathbf{x}_{GS}$ is completely specified once $\mathbf{x}_k$ and the matrix V_0 have been computed. This general solution is required in a number of applications such as total linear least squares (TLLS) [10, § 12.3], solution of nonlinear equations by Gauss-Newton with a rank deficient Jacobian [8, § 4.7.5], and the study of Fredholm integral equations of the first kind [6], [12], [13]. These examples satisfy the above assumption that the numerical rank is well defined, and they require the explicit computation of the TSVD solution (and none of the related solutions mentioned in § 1).

3. The rank revealing QR-factorization. The first step of our algorithm is to determine the numerical rank k of the matrix A. Of course, we could compute the complete SVD and examine the singular values. However, this technique is quite expensive, so instead we want to extract the information from a QR-factorization of A. To be precise, we want a QR-factorization of A in the particular form

$$A\Pi = QR = Q\begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{12} \end{bmatrix}\begin{matrix} k = \text{numerical rank} \\ n-k \end{matrix} \tag{3.1}$$

with $\|R_{22}\|_2$ small such that the QR-factorization (3.1) exhibits any numerical rank deficiency in A. This particular form is termed the *rank revealing* QR-*factorization* (RRQR) of A [2]. The problem of computing the RRQR is equivalent to determining a column permutation Π such that $\|R_{22}\|_2$ is small. It is emphasized that the usual column pivoting strategy does not guarantee this [10, p. 167]. Recently, Chan [2] has described an algorithm for choosing a permutation Π that guarantees $\|R_{22}\|_2$ to be small. The permutation turns out to be identical to one proposed by Foster [7] for subset selection. In addition, Chan's algorithm also gives upper and lower bounds for the singular values of A plus an approximate null space.

To motivate the strategy used in [2], consider the following lemma.

LEMMA 3.1. *Given any column permutation* Π *and an* $X \in \mathbb{R}^{n\times(n-k)}$ *such that*

$$\|AX\|_2 = \varepsilon, \tag{3.2}$$

the QR-*factorization* (3.1) *of* $A\,\Pi$ *yields an* R_{22} *that satisfies*

$$\|R_{22}\|_2 \leqq \varepsilon \|Y_2^{-1}\|_2, \tag{3.3}$$

where the matrix Y_2 *is defined by*

$$\Pi^T X = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix}\begin{matrix} k \\ n-k \end{matrix}. \tag{3.4}$$

Proof. Using (3.1), (3.2), and (3.4), we immediately obtain

$$\varepsilon = \|AX\|_2 = \|R\Pi^T X\|_2 \geqq \|R_{22}Y_2\|_2 \geqq \|R_{22}\|_2 \|Y_2^{-1}\|_2^{-1} \Rightarrow \|R_{22}\|_2 \leqq \varepsilon \|Y_2^{-1}\|_2. \qquad \square$$

The above lemma shows that if we can identify an approximate null space of A of dimension $n-k$, represented by a matrix X that gives a small ε in (3.2), then the near rank-deficiency of A is revealed by a small $R_{22} \in \mathbb{R}^{(n-k)\times(n-k)}$ in the QR-factorization (3.1), provided that we use a permutation Π such that $\|Y_2^{-1}\|_2$ is not large. The strategy in computing the RRQR is therefore to find a matrix X such that ε is small and to find a permutation Π of the rows of X such that $\|Y_2^{-1}\|_2$ is as small as possible. For the special case $k = n-1$, one can take $X = \mathbf{v}_n$ and Π the permutation that brings the largest element in absolute value of $\mathbf{v}_n$ to the last position [9]. The vector $\mathbf{v}_n$ can be estimated as in the LINPACK condition estimator [5] or variants thereof [4], or computed by inverse iteration [16]. In the general case $k \leqq n-1$, one approach is to take $X = V_0$ giving $\varepsilon = \sigma_{k+1}$ and Π determined in such a way that the bottom $(n-k)\times(n-k)$ submatrix of $\Pi^T V_0$ is as well conditioned as possible, as proposed in [9]. The idea of the RRQR algorithm in [2] is to devise a more efficient algorithm based on the QR-factorization by repeated application of the $k=n-1$ procedure. Essentially, the matrices X and Π are constructed from the right singular vectors corresponding to the smallest singular value of appropriately chosen, increasingly smaller subsets of columns of A.

```
ALGORITHM RRQR:
1. Compute any QR-factorization of $A: A\Pi = QR$ and set $k \leftarrow n$.
2. Loop to identify the small singular values of $A$.
    2a. Partition $R = \begin{array}{c|c} \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix} & \begin{matrix} k \\ n-k. \end{matrix} \end{array}$
    2b. Get an initial guess $\delta_k$ of the smallest singular value and the corresponding
        right singular vector $\bar{\mathbf{w}}_k$ of $R_{11}$, e.g., by using the LINPACK condition estimator
        [5] or variants thereof [4].
    2c. If the initial guess $\delta_k$ is numerically nonzero, then improve $\delta_k$ and $\bar{\mathbf{w}}_k$ by
        means of inverse iteration applied implicitly to $R_{11}^T R_{11}$.
    2d. If $\delta_k > \tau$ then $k$ is the numerical rank of $A$ (see (3.6) below): goto 3.
    2e. Determine a permutation $\bar{P}$ such that $|(\bar{P}^T \bar{\mathbf{w}}_k)_k| = \|\bar{P}^T \bar{\mathbf{w}}_k\|_\infty$.
    2f. Compute the QR-factorization: $R_{11}\bar{P} = \bar{Q}\bar{R}_{11}$.
    2g. Update $\Pi$, $Q$, and $R$: $\Pi \leftarrow \Pi \begin{bmatrix} \bar{P} & 0 \\ 0 & I_{n-k} \end{bmatrix}$, $Q \leftarrow Q \begin{bmatrix} \bar{Q} & 0 \\ 0 & I_{n-k} \end{bmatrix}$, $R \leftarrow \begin{bmatrix} \bar{R}_{11} & \bar{Q}^T R_{12} \\ 0 & R_{22} \end{bmatrix}$.
    2h. Assign $\begin{bmatrix} \bar{\mathbf{w}}_k \\ 0 \end{bmatrix}$ to the $k$th column of $W$, and compute: $W \leftarrow \begin{bmatrix} \bar{P}^T & 0 \\ 0 & I_{n-k} \end{bmatrix} W$.
    2i. Let $k \leftarrow k-1$.
3. End.
```

In step 2c we accept a singular value as numerically nonzero if it exceeds $n\varepsilon_M\sigma_1$, where ε_M is the machine precision. Steps 2e and 2f should be implemented as described by [7] and [2] to reduce the computational effort to $O(\frac{1}{2}(n-k)n^2)$ flops. The dominating effort is therefore the backsubstitutions in step 2c, which means that the numerical rank k of A can be determined in $O(p_1(n-k)n^2)$ flops, where p_1 is the average number of iterations used.

From the definition of $\bar{\mathbf{w}}_i$ in steps 2b-2c of RRQR, it follows that Π and W satisfy

$$\|A\Pi W\|_F^2 = \sum_{i=k+1}^{n} \delta_i^2. \tag{3.5}$$

Since the δ_i's are the smallest singular values of increasingly smaller *subsets* of columns of A, it immediately follows from the interlacing inequalities for singular values [10, Cor. 8.3-3] that the δ_i are nonincreasing for increasing values of i and that

$$\delta_i \leqq \sigma_i, \qquad k \leqq i \leqq n. \tag{3.6}$$

Therefore, the RRQR algorithm can be viewed as a method for identifying an approximate null space of A represented by the matrix $X = \Pi W$.

The following theorem gives upper and lower bounds on the smallest singular values of A, which are by-products of algorithm RRQR.

THEOREM 3.2. *Let the quantities δ_i be computed by means of Algorithm RRQR above. Also, let the matrix $W \in \mathbb{R}^{n\times(n-k)}$ be partitioned as*

$$W = \begin{array}{c|c} \begin{bmatrix} W_1 \\ W_2 \end{bmatrix} & \begin{matrix} k \\ n-k \end{matrix} \end{array} \tag{3.7}$$

and let W_2^i and R_{22}^i denote the bottom $(n-i)\times(n-i)$-blocks of W_2 and R_{22} for $k \leqq i \leqq n-1$, respectively. Then

$$\frac{\sigma_{i+1}}{\sqrt{n-i}\|(W_2^i)^{-1}\|_2} \leqq \delta_{i+1} \leqq \sigma_{i+1} \leqq \|R_{22}^i\|_2 \leqq \sigma_{i+1}\sqrt{n-i}\|(W_2^i)^{-1}\|_2. \tag{3.8}$$

Also, we have the a priori bound:

$$\|(W_2^i)^{-1}\|_2 < \sqrt{n}2^{n-i}. \tag{3.9}$$

Proof. We shall prove each of the four inequalities in (3.8), starting from the right. The first follows from Lemma 3.1 if we let $\Pi^T X = W^i$ consist of the last $n-i$ columns of W such that $Y_2 = W_2^i$ and $\varepsilon = \|AX\|_2 = \|A\Pi W^i\|_2 \leqq \|A\Pi W^i\|_F \leqq \sqrt{n-i}\delta_{i+1} \leqq \sqrt{n-i}\sigma_{i+1}$ (where we have used (3.5) and (3.6)). The second follows from the perturbation theorem for singular values [10, Cor. 8.3-2], and the next one is simply (3.6). The last one follows from the second inequality from the right in (3.8) and the relation immediately above.

To prove (3.9) we use the same technique as in [7], but here applying the norm $\|\cdot\|_2$. Define $Z = W_2^i$. Due to the construction of W, it has the following properties: its columns have unit norm; it is upper trapezoidal; and the maximum element (in modulus) in each column is on the diagonal of W_2. If $\eta = n-i$, then we have

$$n^{-1/2} \leqq |z_{jj}| \leqq 1, \qquad 1 \leqq j \leqq \eta,$$
$$0 \leqq |z_{j\zeta}| \leqq n^{-1/2}, \qquad 1 \leqq j, \zeta \leqq \eta, \qquad j \neq \zeta.$$

Now let $\mathbf{y} = Z\mathbf{x}$ with $\|\mathbf{y}\|_2 = 1 \Rightarrow |y_j| \leqq 1$. Then, from the equation

$$x_j = z_{jj}^{-1}\left(y_j - \sum_{\zeta=j+1}^{\eta} z_{j\zeta}x_\zeta\right), \qquad 1 \leqq j \leqq \eta,$$

we get

$$|x_j| \leqq \sqrt{n} + \sum_{\zeta=j+1}^{\eta} |x_\zeta|, \qquad 1 \leqq j \leqq \eta.$$

It is easy to show by induction that $|x_j| \leqq \sqrt{n}2^{\eta-j}$, so that

$$\|\mathbf{x}\|_2^2 = n[4^0 + 4^1 + \cdots + 4^{\eta-1}] = n\frac{4^\eta - 1}{\eta - 1} < n4^\eta$$

and therefore $\|x\|_2 < \sqrt{n}2^\eta$. From the definition of the matrix 2-norm it then follows that

$$\|(W_2^i)^{-1}\|_2 = \|Z^{-1}\|_2 = \min_{\mathbf{x}} \frac{\|\mathbf{x}\|_2}{\|Z\mathbf{x}\|_2} = \min_{\mathbf{x}} \frac{\|\mathbf{x}\|_2}{\|\mathbf{y}\|_2} = \min_{\mathbf{x}} \|\mathbf{x}\|_2 < \sqrt{n}2^\eta. \qquad \square$$

Theorem 3.2 implies that, as long as the numerical rank $k \approx n$ such that the term 2^{n-i} in (3.9) is small, the quantities δ_i and $\|R_{22}^i\|_2$ are guaranteed to be tight upper and lower bounds of the singular values σ_i. This means that Algorithm RRQR will determine the correct numerical rank k as long as there is a well-defined gap between singular values σ_k and σ_{k+1}.

4. Determination of the numerical null space. Algorithm RRQR gives the numerical rank k and an approximate null space, represented by the matrix W. However, the TSVD solution requires the true numerical null space N_k of A. In this section, we show how to refine matrix W to obtain an accurate basis for N_k. Before we give the refinement procedure, we first study how good an approximation W is. From (3.5) and (3.6) it immediately follows that

$$\|A\Pi W\|_2^2 \leqq \|A\Pi W\|_F^2 = \sum_{i=k+1}^{n} \delta_i^2 \leqq \sum_{i=k+1}^{n} \sigma_i^2 \leqq (n-k)\sigma_{k+1}^2,$$

and therefore

$$\frac{\|A\Pi W\|_2}{\|A\|_2} \leqq \sqrt{n-k}\,\omega_k, \tag{4.1}$$

where ω_k is defined in (2.2a). This result suggests that the range of ΠW is a good approximation to the numerical null space N_k, provided that ω_k is small. That this is true follows from the following theorem.

THEOREM 4.1. *Let the matrix W be partitioned as in* (3.7). *Then the subspace angle θ between the range of ΠW and the numerical null space N_k of A is bounded as*

$$\sin\theta \leqq (1+\sqrt{n-k}\|W_2^{-1}\|_2)\omega_k, \tag{4.2}$$

where $\omega_k = \sigma_{k+1}/\sigma_k$, and where $\|W_2^{-1}\|_2 \leqq \sqrt{n}2^{n-k}$ due to (3.9).

Proof. First, we define a QR-factorization of $\Pi W = \tilde{V}_0 R_W$ and obtain the bound

$$\|R_W^{-1}\|_2 = \|W^+\Pi^T\tilde{V}_0\|_2 \leqq \|W^+\|_2 = \sigma_{\min}(W)^{-1} \leqq \sigma_{\min}(W_2)^{-1} = \|W_2^{-1}\|_2. \tag{4.3}$$

In terms of the SVD (1.2) of A, we then define the matrices

$$\begin{aligned} &\Sigma_k = \operatorname{diag}(\sigma_1, \cdots, \sigma_k), && \Sigma_0 = \operatorname{diag}(\sigma_{k+1}, \cdots, \sigma_n), \\ &U_k = [u_1 \cdots u_k], && U_0 = [u_{k+1} \cdots u_n], \\ &V_k = [v_1 \cdots v_k], && V_0 = [v_{k+1} \cdots v_n] \end{aligned} \tag{4.4}$$

and write $A\tilde{V}_0 = U_k\Sigma_k V_k^T\tilde{V}_0 + U_0\Sigma_0 V_0^T\tilde{V}_0$. Then, from [10, Cor. 2.4-2]:

$$\begin{aligned}\sin\theta &= \|V_k^T\tilde{V}_0\|_2 = \|\Sigma_k^{-1}U_k^TU_k\Sigma_kV_k^T\tilde{V}_0\|_2 \\ &\leqq \|\Sigma_k^{-1}\|_2\|U_k^T\|_2\|U_k\Sigma_kV_k^T\tilde{V}_0\|_2 = \sigma_k^{-1}\|U_k\Sigma_kV_k^T\tilde{V}_0\|_2 \\ &= \sigma_k^{-1}\|A\tilde{V}_0 - U_0\Sigma_0V_0^T\tilde{V}_0\|_2 \leqq \sigma_k^{-1}(\|U_0\Sigma_0V_0^T\tilde{V}_0\|_2 + \|A\tilde{V}_0\|_2) \\ &\leqq \sigma_k^{-1}(\|U_0\|_2\|\Sigma_0\|_2\|V_0^T\tilde{V}_0\|_2 + \|A\tilde{V}_0R_WR_W^{-1}\|_2) \\ &\leqq \sigma_k^{-1}(\sigma_{k+1} + \|A\Pi W\|_2\|R_W^{-1}\|_2) \leqq \sigma_k^{-1}(\sigma_{k+1} + \sqrt{n-k}\sigma_{k+1}\|W_2^{-1}\|_2) \\ &\leqq (1+\sqrt{n-k}\|W_2^{-1}\|_2)\omega_k.\end{aligned}$$

Here we have used (4.3) and $\|V_0^T\tilde{V}_0\| \leqq 1$. □

Theorem 4.1 means that if ω_k is small (corresponding to a large gap in the singular value spectrum of A) and $k \approx n$, then the range of ΠW is a good approximation to the numerical null space N_k, since $\|W_2^{-1}\|_2$ cannot be large, cf. (3.9). Hence, W is a good starting matrix for simultaneous inverse iteration with R^TR in order to determine an accurate basis for N_k, guaranteeing fast convergence within a few steps. This leads to the following algorithm SPIT (subspace iteration), based on an algorithm in [3], to determine a matrix $\bar{V}_0$ whose columns are orthonormal basis vectors for N_k.

```
ALGORITHM SPIT:
1. Let $i \leftarrow 0$.
2. Set $\bar{V}^{(0)} \leftarrow W$ and orthonormalize its columns.
3. Subspace iteration. Repeat until $\sin\psi < \varepsilon$:
    3a. $i \leftarrow i+1$, $\bar{U}^{(i)} \leftarrow R^{-T}\bar{V}^{(i-1)}$, $\bar{V}^{(i)} \leftarrow R^{-1}\bar{U}^{(i)}$. Scaling is necessary to avoid overflow!
    3b. Orthonormalize the columns of $\bar{V}^{(i)}$.
```

```
3c. Estimate $\sin\psi$, where $\psi$ is the subspace angle between range$(\bar{V}^{(i-1)})$ and range$(\bar{V}^{(i)})$.
4. Orthonormalize the columns of $\bar{U}^{(i)}$ to get $\bar{U}_0$.
5. Let $\bar{V}_0 \leftarrow \Pi \bar{V}^{(i)}$.
6. End.
```

If R has zero or nearly zero diagonal elements, simply insert a small multiple of the machine precision. The quantity ε in step 3 determines the accuracy of the computed $\bar{V}_0$, and should be set according to the required accuracy of the computed TSVD solution x_k. In step 3c, notice that $\sin\psi$ is given by

$$\sin\psi \equiv \|(I_n - \bar{V}^{(i)}\bar{V}^{(i)T})\bar{V}^{(i-1)}\|_2 = \|\bar{P}^{(i)}\bar{V}^{(i-1)}\|_2, \tag{4.5}$$

in which $\bar{P}^{(i)}$ is the projection matrix for orthogonal projection onto the orthogonal complement of range$(\bar{V}^{(i)})$. Hence, the matrix $\bar{P}^{(i)}\bar{V}^{(i-1)}$ can be computed simply by orthonormalizing the columns of $\bar{V}^{(i-1)}$ with respect to the columns of $\bar{V}^{(i)}$ (e.g., by the modified Gram-Schmidt procedure). The quantity $\sin\psi$ can then easily be estimated by a simple estimate of $\|\bar{P}^{(i)}\bar{V}^{(i-1)}\|_2$. The computational effort is therefore dominated by the subspace iterations in step 3a. Hence, $\bar{V}_0$ is determined in $O(p_2(n-k)n^2)$ flops, where p_2 is the number of iterations.

5. Determination of the TSVD solution. The remaining problem now is to compute the TSVD solution $\mathbf{x}_k$ (2.3) from the quantities already obtained from RRQR and SPIT. Suppose that the matrix R in (3.1) has no exact zero singular values. In theory, one could then compute the vector $R^{-1}Q^T\mathbf{b}$, which is a member of the general solution $\mathbf{x}_{GS}$ in (2.3), and then deflate this solution; i.e., orthogonalize with respect to the numerical null space N_k. In practice, however, the matrix R may be highly ill-conditioned which will completely destroy the accuracy of the computed TSVD solution. This is because the *computed* vector $R^{-1}Q^T\mathbf{b}$, which theoretically should be given by (1.3), will be dominated by those contributions, corresponding to the small singular values $\sigma_{k+1}, \cdots, \sigma_n$, that are to be removed again in the deflation process. Therefore, the rounding errors associated with this component will contaminate the computed solution.

The problem is overcome by also deflating the right-hand side $\mathbf{b}$ with respect to the subspace span$\{\mathbf{u}_{k+1}, \cdots, \mathbf{u}_n\}$, where $\mathbf{u}_i$ are the columns of U in the SVD (1.2) of A. This idea is similar to those used in [1] and [16]. To deflate with respect to the left singular vectors $\{\mathbf{u}_{k+1}, \cdots, \mathbf{u}_n\}$, deflate $Q^T\mathbf{b}$ with respect to the columns of $\bar{U}_0$ provided by step 4 of SPIT. The algorithm TSOL for computing the TSVD solution x_k then becomes

```
ALGORITHM TSOL:
1. Compute $\boldsymbol{\beta} \leftarrow Q^T\mathbf{b}$.
2. Compute $\boldsymbol{\xi} \leftarrow R^{-1}(I_n - \bar{U}_0\bar{U}_0^T)\boldsymbol{\beta}$, where $\bar{U}_0$ is defined in 4 of SPIT.
3. Compute the TSVD solution $\mathbf{x}_k = (I_n - \bar{V}_0\bar{V}_0^T)\Pi\boldsymbol{\xi}$, where $\bar{V}_0$ is defined in step 5 of SPIT.
4. End.
```

This algorithm involves one backsubstitution such that $\mathbf{x}_k$ is determined in $\frac{1}{2}n^2$ flops. Notice that if A has q numerically zero singular values, then the corresponding left singular vectors of A are given with sufficient accuracy by

$$[\mathbf{u}_{n-q+1} \cdots \mathbf{u}_n] = Q\begin{bmatrix} 0 \\ I_q \end{bmatrix}\begin{matrix} n-q \\ q \end{matrix}, \tag{5.1}$$

which means that deflation with respect to the associated subspace simply corresponds to neglection of the last q elements of $Q^T\mathbf{b}$.

Summarizing, the operations count for computing the TSVD solution $\mathbf{x}_k$ via RRQR, SPIT, and TSOL can be shown to be bounded by

$$[m+(n-k)(p_1+p_2+\tfrac{1}{2})+p_1+\tfrac{1}{2}]n^2 \quad \text{flops,} \tag{5.2}$$

where p_1 is the average number of inverse iterations per singular value in RRQR, and p_2 is the number of inverse subspace iterations used in SPIT. Thus, if the numerical rank $k \approx n$, then the RRQR-SPIT-TSOL procedure does not cost much more computational effort than the cost of computing one QR-factorization.

6. Extensions. Although the above algorithm is designed for problems with $m \geqq n$, it can easily be extended to the underdetermined case $B \in R^{m\times n}$ with $m<n$. In this case, first compute a QR-factorization of B^T:

$$B^T=[Q_1, Q_2]\begin{bmatrix} R_1 \\ 0 \end{bmatrix}\begin{matrix} m \\ n-m \end{matrix} \tag{6.1}$$

and then apply our algorithm to $A=R_1^T$. Let $\mathbf{x}_k^{(B)}$, $\bar{V}_0^{(B)}$ and $\mathbf{x}_k^{(A)}$, $\bar{V}_0^{(A)}$ represent the general solutions associated with B and $A=R_1^T$, respectively. Obviously, the TSVD solution $\mathbf{x}_k^{(B)}$ lies in the range of Q_1, and the columns of Q_2 span an exact null space of B. Hence, $\mathbf{x}_k^{(B)}$ and $\bar{V}_0^{(B)}$ are obtained from the results $\mathbf{x}_k^{(A)}$ and $\bar{V}_0^{(A)}$ of the RRQR-SPIT-TSOL algorithm by setting

$$\mathbf{x}_k^{(B)} \leftarrow Q_1\mathbf{x}_k^{(A)}, \qquad \bar{V}_0^{(B)} \leftarrow [\bar{V}_0^{(A)}, Q_2]. \tag{6.2}$$

We notice that our algorithm may also be used for subset selection as described by Golub, Klema, and Stewart [9]. In their method, the column pivoting for subset selection is determined from examining the matrix V_0 in (4.4) obtained from a complete SVD of A. Since this matrix V_0 and the matrix $\bar{V}_0$ computed by SPIT span the same subspace N_k, we can apply the same procedure to $\bar{V}_0$ and obtain a similar subset selection procedure. This permutation is, in general, different from the one obtained from the procedure in [9] as well as the permutation provided by the method of Foster [7]. However, they should all produce residuals of the same size.

Finally, we stress that we have made no a priori assumption about the pivoting in the initial QR-factorization of A. Hence, if A is sparse, this QR-factorization can be computed with pivoting for sparsity. Also, the QR-factorization in step 2f of RRQR can be performed by taking advantage of the sparsity of R as described by Heath [14]. This means that our algorithm is also suited for sparse matrices as long as the numerical rank $k \approx n$ (since the matrices $\bar{U}^{(i)}$ and $\bar{V}^{(i)}$ are *full* matrices).

7. Numerical examples. In this section we give a few illustrative examples of the application of our algorithm. The first four examples are constructed to illustrate the features and properties of the algorithm; the fifth example is a practical example from the application of integral equations in two-dimensional potential theory.

For examples 1-4, we generated four small matrices of dimensions $m=25$, $n=10$ with numerical rank $k=7$ and different values of ω_k. All matrices were generated by replacing the singular values in the SVD of randomly generated matrices. The first seven singular values are fixed at

$$\sigma_1=1, \quad \sigma_2=0.5, \quad \sigma_3=0.2, \quad \sigma_4=0.1, \quad \sigma_5=0.05, \quad \sigma_6=0.02, \quad \sigma_7=0.01, \tag{7.1}$$

whereas the last singular values σ_8, σ_9, and σ_{10} are varied. In Table 1 we tabulate, for each of the small singular values σ_i, the upper and lower bounds δ_i and $\|R_{22}^i\|_2$ from Algorithm RRQR and the number of inverse iterations used in RRQR to obtain it.

TABLE 1
Comparison of the smallest singular values of A with the upper and lower bounds from RRQR. *The number of iterations used in* RRQR *is also given.*

Example	i	δ_i	σ_i	$\|R_{22}^i\|_2$	RRQR iterations
1	7	$8.28\cdot10^{-3}$	$1.00\cdot10^{-2}$	$2.08\cdot10^{-2}$	3
	8	$5.27\cdot10^{-18}$	$9.71\cdot10^{-18}$	$1.89\cdot10^{-17}$	2
	9	$5.85\cdot10^{-18}$	$7.38\cdot10^{-18}$	$1.43\cdot10^{-17}$	0
	10	$8.95\cdot10^{-18}$	$4.48\cdot10^{-18}$	$9.87\cdot10^{-18}$	0
2	7	$7.79\cdot10^{-3}$	$1.00\cdot10^{-2}$	$2.14\cdot10^{-2}$	5
	8	$9.86\cdot10^{-6}$	$1.00\cdot10^{-5}$	$1.92\cdot10^{-5}$	2
	9	$9.15\cdot10^{-7}$	$1.00\cdot10^{-6}$	$1.81\cdot10^{-6}$	2
	10	$1.00\cdot10^{-7}$	$1.00\cdot10^{-7}$	$1.75\cdot10^{-7}$	2
3	7	$7.81\cdot10^{-3}$	$1.00\cdot10^{-2}$	$2.14\cdot10^{-2}$	5
	8	$9.86\cdot10^{-4}$	$1.00\cdot10^{-3}$	$1.91\cdot10^{-3}$	2
	9	$9.15\cdot10^{-5}$	$1.00\cdot10^{-4}$	$1.81\cdot10^{-4}$	2
	10	$1.00\cdot10^{-5}$	$1.00\cdot10^{-5}$	$1.75\cdot10^{-5}$	2
4	7	$8.30\cdot10^{-3}$	$1.00\cdot10^{-2}$	$2.15\cdot10^{-2}$	5
	8	$4.94\cdot10^{-3}$	$5.00\cdot10^{-3}$	$9.00\cdot10^{-3}$	3
	9	$1.87\cdot10^{-3}$	$2.00\cdot10^{-3}$	$3.60\cdot10^{-3}$	2
	10	$1.00\cdot10^{-3}$	$1.00\cdot10^{-3}$	$1.71\cdot10^{-3}$	2

Consider first the quality of the bounds on the singular values. In all four examples, these bounds are very tight, and the numerical rank k is easily identified as $k=7$. Moreover, the number of inverse iterations needed is very small (although the number of iterations required to estimate $\sigma_k=\sigma_7$ increases with increasing ω_k).

In Table 2 we tabulate results associated with the null-space computation and the TSVD solutions. The first part of the table compares the angle θ between the approximate null space, as computed by RRQR, and the exact N_k, with the bound given in (4.2). Notice that the columns of the matrix ΠW indeed give an approximate basis

TABLE 2
Results associated with the null-space computation: ω_k, *the bound in* (4.2), *the angle θ between* range (ΠW) *and* N_k, *the number of iterations in* SPIT, *and the angle θ between* range $(\bar V_0)$ *and* N_k. *Also shown are the relative errors in* $\bar x_k$ *and* $\bar{\bar x}_k$.

Example	1	2	3	4
$\omega_k=\sigma_{k+1}/\sigma_k$	$9.71\cdot10^{-16}$	$1.00\cdot10^{-3}$	$1.00\cdot10^{-1}$	$5.00\cdot10^{-1}$
$(1+\sqrt{n}\|W_2^{-1}\|_2)\omega_k$	$9.63\cdot10^{-15}$	$5.05\cdot10^{-3}$	$5.00\cdot10^{-1}$	$2.52\cdot10^{0}$
$\sin\theta$ {range (ΠW), N_k}	$7.43\cdot10^{-16}$	$4.27\cdot10^{-8}$	$4.28\cdot10^{-4}$	$1.15\cdot10^{-2}$
Iterations in SPIT	1	2	5	14
$\sin\theta$ {range $(\bar V_0)$, N_k}	$6.05\cdot10^{-16}$	$1.20\cdot10^{-15}$	$1.91\cdot10^{-14}$	$2.44\cdot10^{-11}$
$\dfrac{\|\mathbf{x}_k-\bar{\mathbf{x}}_k\|_2}{\|\mathbf{x}_k\|_2}$	$5.45\cdot10^{-15}$	$1.22\cdot10^{-15}$	$4.78\cdot10^{-11}$	$4.78\cdot10^{-11}$
$\dfrac{\|\mathbf{x}_k-\bar{\bar{\mathbf{x}}}_k\|}{\|\mathbf{x}_k\|_2}$	$8.70\cdot10^{-2}$	$3.01\cdot10^{-12}$	$4.78\cdot10^{-11}$	$4.78\cdot10^{-11}$

for N_k, and that the quality (as expected) depends on the quantity ω_k. The upper bound for $\sin\theta$ is overly pessimistic, but it indicates the general behavior of $\sin\theta$.

The second part of Table 2 shows the performance of Algorithm SPIT. We see that as long as there is a large gap between σ_k and σ_{k+1} then the simultaneous inverse iterations converge fast. In example 4, the gap is very small, and the iterations converge very slowly. The accuracy of the computed $\bar{V}_0$ reflects the stopping criteria $\varepsilon = 10^{-10}$ used in step 3 of SPIT.

In the last part of Table 2 we give the results from solving the least squares problem (1.1) by means of Algorithm TSOL. Right-hand sides $\mathbf{b}$ were generated by the formula

$$\mathbf{b} = \sum_{i=1}^{m} \mathbf{u}_i \tag{7.2}$$

such that $\mathbf{b}$ has large components in both the range of A_k and its orthogonal complement. We computed two solutions:

$\tilde{\mathbf{x}}_k$: computed by TSOL,

$\tilde{\tilde{\mathbf{x}}}_k$: computed by TSOL *without* deflation of the right-hand side $\mathbf{b}$.

These two solutions are compared with the true solution $\mathbf{x}_k$. We see that the relative accuracy of $\tilde{\mathbf{x}}_k$ is the same as the accuracy of $\bar{V}_0$. We also see that deflation of $\mathbf{b}$ is in fact necessary when very small singular values are present in the matrix R.

The conclusion to be drawn from these numerical examples is that our algorithm performs well as long as the assumption $\omega_k \ll 1$ is satisfied. In fact, even when ω_k is not small, our algorithm is able to give good results at the cost of the larger computational effort involved in the large number of simultaneous inverse iterations.

The practical example comes from the numerical analysis of linear algebraic equations derived from first kind integral equations in two-dimensional potential theory [12]. In particular, we consider the homogeneous Fredholm integral equations of the first kind:

$$\int_0^2 \ln|s-x| f(s)\, ds = 0, \qquad x \in [0,2]. \tag{7.3}$$

The solution to this equation has a physical interpretation, for example, as the steady-state charge distribution on a line segment form $s=-2$ to $s=2$. Due to symmetry it is only necessary to consider the interval $x \in [0,2]$. Choosing $m = n = 10$ and discretizing (7.3) as described in [12], we are led to a matrix A, which is then processed by RRQR and SPIT, and the results are shown in Table 3. The results *guarantee* that the numerical rank of A is $k=9$ (i.e., *one* small singular value). The solution $f(s)$ to (7.3) can then be computed from the numerical null space of A; i.e., from $\bar{V}_0 \in \mathbb{R}^{n\times 1}$ produced by SPIT, and the such computed $f(s)$ agrees with the true TSVD solution as computed in [12]. Hence, we have shown that the study of the integral equation (7.3) can be performed perfectly well by means of our algorithm, thus avoiding the large computational effort of computing the complete SVD of A.

TABLE 3
Results associated with the discretization of the integral equation (7.3)

Example	i	δ_i	σ_i	$\|R_{22}^i\|_2$	RRQR iterations
5	1	—	$2.09\cdot 10^{0}$	—	—
	9	$4.60\cdot 10^{-2}$	$6.01\cdot 10^{-2}$	$6.33\cdot 10^{-2}$	2
	10	$2.61\cdot 10^{-6}$	$2.61\cdot 10^{-6}$	$3.72\cdot 10^{-6}$	2

8. Conclusion. The TSVD solution to a rank deficient least squares problem usually requires computation of the singular value decomposition (SVD) of the matrix, which involves a large amount of computational effort compared to a QR-factorization. In this paper we have shown that the TSVD solution can be computed much more efficiently from a rank revealing QR-factorization (RRQR) of the matrix, followed by inverse subspace iterations to improve the estimated null space of the matrix. Unlike methods based on heuristic procedures for rank determination, our algorithm is guaranteed to perform reliably when the matrix has a well-determined numerical rank, as is the case in a number of practical applications. This suggests the TSVD method as a favorable alternative to regularization for such problems.

Acknowledgments. The authors would like to thank Professor Gene Golub for his hospitality during our visits to the Department of Computer Science at Stanford University, and the referee for several useful comments to the manuscript.

REFERENCES

[1] T. F. CHAN, *Deflated decomposition of solutions of nearly singular systems*, SIAM J. Numer. Anal., 21 (1984), pp. 738-754.

[2] ———, *Rank revealing* QR-*factorizations*, Linear Algebra Appl., 88/89, pp. 67-82.

[3] T. F. CHAN AND D. C. RESASCO, *Generalized deflated block-elimination*, SIAM J. Numer. Anal., 23 (1986), pp. 913-924.

[4] A. K. CLINE, A. R. CONN, AND C. VAN LOAN, *Generalizing the* LINPACK *condition estimator*, in Numerical Analysis, J.P. Hennart, ed., Lecture Notes in Mathematics 909, Springer-Verlag, Berlin, New York, 1982.

[5] J. J. DONGARRA, J. R. BUNCH, C. B. MOLER, AND G. W. STEWART, LINPACK *Users Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

[6] L. ELDÉN, *The numerical solution of a non-characteristic Cauchy problem for a parabolic equation*, in Numerical Treatment of Inverse Problems in Differential and Integral Equations, P. Deuflhard and E. Hairer, eds., Birkhäuser, Basel, 1983.

[7] L. V. FOSTER, *Rank and null space calculations using matrix decomposition without column interchanges*, Linear Algebra Appl., 74 (1986), pp. 47-71.

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

[9] G. H. GOLUB, V. KLEMA, AND G. W. STEWART, *Rank degeneracy and least squares problems*, Tech. Report TR-456, Department of Computer Science, University of Maryland, College Park, MD, 1976.

[10] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, North Oxford Academic, New York, 1983.

[11] P. C. HANSEN, *The truncated* SVD *as a method for regularization*, BIT, 27 (1987), pp. 534-553.

[12] ———, *Solution of ill-posed problems by means of truncated* SVD, in Numerical Mathematics, Singapore 1988, ISNM 86, R. P. AGARWAL, Y. M. CHOW, AND S. F. WILSON, eds., Birkhäuser, 1988.

[13] P. C. HANSEN AND S. CHRISTIANSEN, *An* SVD *analysis of linear algebraic equations derived from first kind integral equations*, J. Comput. Appl. Math., 12/13 (1985), pp. 341-357.

[14] M. T. HEATH, *Some extensions of an algorithm for sparse linear least squares problems*, SIAM J. Sci. Statist. Comput., 3 (1982), pp. 223-237.

[15] C. L. LAWSON AND R. J. HANSON, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, N.J., 1974.

[16] G. W. STEWART, *On the implicit deflation of nearly singular systems of linear equations*, SIAM J. Sci. Statist. Comput., 2 (1981), pp. 136-140.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 3, pp. 531–544, May 1990

THE PROBABILITY OF LARGE DIAGONAL ELEMENTS IN THE *QR* FACTORIZATION*

LESLIE V. FOSTER†

Abstract. In theory the minimum diagonal element d in a QR factorization of an $m \times n$ matrix A can be much larger than the smallest singular value s_n of A. However, in practice d is often the same magnitude as s_n and some packages rely on this to test for approximate singularity. We develop sufficient conditions and separate necessary conditions, involving A's nth singular vector, for $d >> s_n$. For a natural class of random matrices these conditions are used to obtain bounds on the probability that $d/s_n \geq B$. In certain cases for large B this probability is not insignificant. This suggests that tests for approximate singularity which are based on the size of d should be used with caution. In estimating some of our probabilities we introduce techniques that lead to an enormous reduction in the required sample size.

Key words. QR-decomposition, rank deficiency, condition

AMS(MOS) subject classification. 15, 65F

1. Introduction.

The detection of rank deficiency or near rank deficiency in a linear system of equations is important in many applications [AG], [AH], [E], [EM], [HaC], [Han], [Ke], [Ka], [LH], [LN], [ML], [MR], [N], [P], [TA], [S1], [TK], [VD], [V]. Consequently, there is a variety of techniques including the singular value decomposition [GVL] and condition estimators [CMWS], [CR], [ZWS] that can be used in practice to detect near rank deficiency. One of the simpler tools that has sometimes been successfully used to detect near rank deficiency in an $m \times n$ matrix A, with $m \geq n$, is the following: form a QR factorization $A = QR$ of A, where Q is orthogonal and R is upper right triangular and examine the diagonal entries of R for "small" entries. In this paper we will examine conditions under which this simple scheme will work and, for certain classes of matrices A, determine the probability that it fails.

Our motivation for this study is twofold. First as noted in [CR] "empirical evidence suggests that there is a large class of matrices whose conditioning is predicted by examination of the diagonal of the upper triangular factor" in a related decomposition. Indeed, based on such empirical evidence for the QR decomposition, the widely distributed SPARSPAK package [B], [Hea], [GN] makes use of the scheme that we outlined above to detect rank deficiencies in least squares problems. Since this scheme is successful often enough to be used in practice, it is important to know when it fails and to have some sense of the probability that it fails.

Our second motivation is more general: there are several important unresolved questions concerning rare phenomena in matrix decompositions. These relate to the numerical stability of the LU decomposition with partial pivoting [Ka], [T] and the reliability of condition estimators such as those based on the QR decomposition with column interchanges [Hi1], [S2], [DBMS, p. 9.25] or on the technique of Cline, Moler, Stewart, and Wilkinson [CMSW]. In our analysis we will explicitly determine the

*Received by the editors March 18, 1987; accepted for publication (in revised form) June 16, 1989. The work was partially supported by the National Science Foundation under grant DMS8505783 and by the San Diego Supercomputer Center.

†Department of Mathematics and Computer Science, San Jose State University, San Jose, California 95192.

probabilities of some rare (we calculate probabilities down to 10^{-16}) events related to the QR decomposition without column interchanges. We hope that some of the ideas we introduce will be helpful in resolving these other questions relating to rare phenomena in matrix decomposition.

One might hope for small diagonal entries in R when A is almost rank deficient, since if A is exactly rank deficient, then R has at least one zero diagonal entry. However, this hope is not always realized, since it is known [C1], [GW] that the minimal magnitude diagonal entry in R, $d = \min_{i=1,\cdots,n} |r_{ii}|$ is bounded by the singular values s_i, $i = 1, \cdots, n$, $s_1 \geq s_2 \geq \cdots \geq s_n$, of A by

$$s_n \leq d \leq s_n (s_1/s_n)^{(1-1/n)} \tag{1.1}$$

and these are, in general, the best possible bounds. For example, the class of matrices illustrated for $n = 3$ by

$$R_3 = \begin{pmatrix} \epsilon & -1 & 0 \\ 0 & \epsilon & -1 \\ 0 & 0 & \epsilon \end{pmatrix}$$

has $d = \epsilon$, $s_1 = 1+O(\epsilon)$, and $s_n = \epsilon^n + o(\epsilon^n)$. Thus for $\epsilon = .1$ and $n = 10$, say, $s_n << d$. Since a small singular value (for $s_1 \cong 1$, as above), is generally considered the best indicator of near rank deficiency [GKS], [GVL], in this example d fails to reflect this near rank deficiency. However, in view of the empirical evidence that usually $d \cong s_n$, it is important to ask under what conditions is d/s_n large and what is the probability that this occurs.

In §2 of this paper we will derive necessary conditions that d/s_n is large and separate sufficient conditions. These conditions involve the components of the nth right singular vector $\mathbf{v}_n$ of A and, to briefly preview one of our results, we show that if d/s_n is sufficiently large, then the last few components of $\mathbf{v}_n$ must be small (see Theorem 2.8). In §3 we examine a natural class of random matrices A and determine upper bounds and estimates of lower bounds on the probability that d/s_n is large. To preview these results we show that in certain cases the probability $P(d/s_n \geq B)$ that $d/s_n \geq B$ is approximately $\sqrt{2n/\pi}/B$. For example, in such cases if $n = 50$, then $P(d/s_n \geq 100) \cong 5.6$ percent and $P(d/s_n \geq 1000) \cong .56$ percent. Therefore, although d is usually of the same magnitude as s_n, the probability that d is substantially larger than s_n is not insignificant. These results suggest that one should be cautious in using d for detection of near rank deficiency. Section 4 of the paper summarizes our results and discusses potential extensions.

Here we might say a few words about our methods in relation to other unresolved questions concerning matrix decomposition. Perhaps the most striking result of our work is that in estimating some probabilities we are able, by use of appropriate necessary conditions, to achieve an enormous reduction in the sample size required. In some cases we reduced this sample size by a factor of 10^{-14}. Perhaps techniques like ours will be useful in resolving other open questions relating to rare events in matrix decomposition.

Literature relating to our results will be discussed in the body of the paper. However, here we should mention that [F] and [C2] present alternatives to the standard QR factorization without column pivoting that can guarantee that $d \cong s_n$ and which incur little extra cost. One motivation for this current work is to demonstrate the need for these new "rank-revealing" QR factorizations. Also we should mention that in [C1] Chan discusses the detection of near rank deficiency by looking for a small diagonal entry in the upper triangular portion of an LU factorization. This is closely related to, although different from our QR-based scheme. Our results are more general than those of Chan in that our results relate to the minimal diagonal entry of the triangular factor, whereas Chan considers only the nth diagonal entry; and we calculate probabilities, whereas Chan does not.

Also, we should mention here that although our results are developed in relation to the QR algorithm without column interchanges they have wider applicability. When applied to AP where P is a permutation matrix, the necessary and sufficient conditions of §2 are applicable to the QR algorithm with column interchanges, $QR = AP$. Also, many of the probabilistic results of §3 are also applicable to the QR algorithm with column interchanges if P is chosen in a manner that is independent of the numerical values in A.

2. Conditions for large diagonal entries. In this section we begin by deriving in Theorem 2.6 upper and lower bounds on the diagonal entries in R in terms of the singular values of A and the components of the nth right singular vector. These bounds will lead to necessary conditions (Theorem 2.8) and sufficient conditions (Theorem 2.11) that d/s_n is large. Later, in §3, we will see that these necessary conditions can be used to obtain an enormous reduction in the sample size required to estimate certain probabilities. The necessary conditions involve the condition number $C = C(A) = s_1/s_n$, of A and our sufficient conditions involve $s_{n-1}/s_n = G(A) = G$ (for gap).

We will need a variety of known results, which we describe first. The singular value decomposition (SVD) of an $m \times n$ matrix A is $A = UDV^T$, where T indicates transpose, U is an $m \times m$ orthogonal matrix, V is an $n \times n$ orthogonal matrix, and D is an $m \times n$ diagonal matrix whose diagonal entries, the singular values of A, are $s_1 \geq s_2 \geq \cdots \geq s_n$ (assuming $m \geq n$). The right (left) singular vectors of A are the columns of $V(U)$. If $\mathbf{x} \in R^n$, Euclidean n dimension space, then we will define and use $\|\mathbf{x}\| = (\sum_{i=1}^n x_i^2)^{1/2}$ and $\|A\| = \max_{\mathbf{x}\neq 0} \|A\mathbf{x}\|/\|\mathbf{x}\|$. If $\mathbf{a}_k \in R^m$ is column k of the $m \times n$ matrix A of full column rank, if the $m \times (k-1)$ matrix A_{k-1} is the first $k-1$ columns of A and if $A = QR$, then it follows easily by standard results [GVL] that the (k, k) component, $r_{kk} = r_{kk}(A)$ of R, satisfies

$$r_{kk}^2 = \min_{\mathbf{x}\in R^{k-1}} \|\mathbf{a}_k - A_{k-1}\mathbf{x}\|^2. \tag{2.1}$$

Furthermore, for such a matrix A there is a unique QR factorization if the diagonal entries of R are selected to be positive [DBMS]. In the following discussion we will use "QR factorization" to denote such a factorization.

We now present three useful lemmas.

LEMMA 2.2. *If D and $\hat{D}$ are $m \times m$ diagonal matrices with positive entries, A is an $m \times n$ matrix of full column rank, $D \geq \hat{D}$, $DA = QR$, and $\hat{D}A = \hat{Q}\hat{R}$, then $r_{kk} \geq \hat{r}_{kk}$, $k = l, \cdots, n$.*

Proof. $r_{kk} = \min_{\mathbf{x}\in R^{k-1}} \|D(\mathbf{a}_k - A_{k-1}\mathbf{x})\|$. Let $\mathbf{x}^*$ be an $\mathbf{x}$ where the minimum is achieved. Then $r_{kk} = \|D(\mathbf{a}_k - A_{k-1}\mathbf{x}^*)\| \geq \|\hat{D}(\mathbf{a}_k - A_{k-1}\mathbf{x}^*)\| \geq \min_{\mathbf{x}\in R^{k-1}} \|\hat{D}(\mathbf{a}_k - A_{k-1})\mathbf{x}\| = \hat{r}_{kk}$. □

LEMMA 2.3. *If an $m \times n$ matrix A has an SVD $A = UDV^T$ and the $m \times n$ matrix $\hat{A}$ has the SVD $\hat{A} = U\hat{D}V^T$ with $D \geq \hat{D} > 0$, then $r_{kk} \geq \hat{r}_{kk}$, $k = 1, 2, \cdots, n$.*

Proof. If $A = QR$ and $\hat{A} = \hat{Q}\hat{R}$, then $DV^T = (U^TQ)R$ and $\hat{D}V^T = (U^T\hat{Q})\hat{R}$. The result then follows from Lemma 2.2. □

LEMMA 2.4. *Suppose the $m \times n$ matrix A, with $m \geq n$, has an SVD, $A = UDV^T$ such that the first $n-1$ singular values of A are identical, $s_1 = s_2 = \cdots = s_{n-1} \geq s_n \neq$*

0 and let $w_k = \sum_{i=k}^n v_{in}^2$, $w_{n+1} = 0$. *Then the diagonal entries of a QR factorization of A satisfy*

$$(2.5) \qquad r_{kk}^2 = s_1^2 \frac{w_{k+1} + (s_n/s_1)^2(1 - w_{k+1})}{w_k + (s_n/s_1)^2(1 - w_k)}, \quad k = 1, 2, \cdots, n.$$

Proof. Let $\mathbf{u}_i \in R^m$, $i = 1, \cdots, n$, and $\mathbf{v}_i \in R^n$, $i = 1, \cdots, n$, respectively, be columns of U and V. (For $i = 1, \cdots, n-1$, $\mathbf{u}_i$ and $\mathbf{v}_i$ are not unique. However, this does not affect the proof.) Note that $A = s_1 \sum_{j=1}^{n-1} \mathbf{u}_j \mathbf{v}_j^T + s_n \mathbf{u}_n \mathbf{v}_n^T$ implies that $A^T A = s_1^2 I + (s_n^2 - s_1^2)\mathbf{v}_n \mathbf{v}_n^T$. Let $\mathbf{v}_n^T \equiv (\mathbf{v}^T, \mu, \mathbf{z}^T)$, where $\mathbf{v} \in R^{k-1}$, and let $\theta = (s_n/s_1)^2 - 1$. Then by (2.1)

$$r_{kk}^2 = \min_{\mathbf{x}} \|A \begin{pmatrix} \mathbf{x} \\ 1 \\ 0 \end{pmatrix}\|^2 = \min_{\mathbf{x}} s_1^2(\mathbf{x}^T(I + \theta \mathbf{v}^T\mathbf{v})\mathbf{x} + 2\theta\mu\mathbf{v}^T\mathbf{x} + \theta\mu^2 + 1).$$

The quadratic function on the right is minimized at

$$\mathbf{x} = -\theta\mu(I + \theta\mathbf{v}\mathbf{v}^T)^{-1}\mathbf{v} = -\theta\mu\mathbf{v}/(1 + \theta\mathbf{v}^T\mathbf{v}).$$

Substituting this value into the quadratic function we get $r_{kk}^2 = s_1^2(1+\theta\mathbf{v}^T\mathbf{v}+\theta\mu^2)/(1+\theta\mathbf{v}^T\mathbf{v})$, which reduces to expression (2.5) using $\mathbf{v}^T\mathbf{v} = 1 - w_k$ and $\mathbf{v}^T\mathbf{v} + \mu^2 = 1 - w_{k+1}$. □

Now we can derive the following very useful bounds on $r_{kk}(A)$.

THEOREM 2.6. *Suppose the* $m \times n$ *matrix A is of full column rank and has an SVD* $A = UDV^T$. *Let* $w_k = \sum_{i=k}^n v_{in}^2$, $w_{n+1} = 0$, $C = s_1/s_n$ *and* $G = s_{n-1}/s_n$. *Then, for* $k = 1, \cdots, n$,

$$\frac{s_{n-1}^2[w_{k+1} + (1 - w_{k+1})/G^2]}{[w_k + (1 - w_k)/G^2]} \le r_{kk}^2(A) \le \frac{s_1^2[w_{k+1} + (1 - w_{k+1})/C^2]}{[w_k + (1 - w_k)/C^2]}.$$

Proof. Let D^* be an $m \times n$ diagonal matrix with diagonal entries $(s_1, \cdots, s_1, s_n)$ and let D_* be an $m \times n$ diagonal matrix with diagonal entries $(s_{n-1}, \cdots, s_{n-1}, s_n)$. The result follows immediately by applying Lemmas 2.3–2.4 to $A_* = UD_*V^T$ and $A^* = UD^*V^T$. □

A valuable corollary follows.

COROLLARY 2.7. *For A,* w_k*, and G as in Theorem* 2.6, *and any* $1 \le k \le n$, *it follows that*

$$s_{n-1}^2[w_{k+1} + (1 - w_{k+1})/G^2] \le r_{jj}^2(A), \qquad j = 1, \cdots, k.$$

Proof. By the definition of w_{j+1}, $w_{k+1} \le w_{j+1}$ for $1 \le j \le k$ and therefore $w_{k+1} + (1 - w_{k+1})/G^2 \le w_{j+1} + (1 - w_{j+1})/G^2$. Furthermore, since $w_j \le 1$ and $G \ge 1$, then $w_j + (1 - w_j)/G^2 \le 1$. Therefore $w_{k+1} + (1 - w_{k+1})\ G^2 \le [w_{j+1} + (1 - w_{j+1})/G^2]/[w_j + (1 - w_j)/G^2]$ and by Theorem 2.6 the result follows. □

We are now ready to state a necessary condition that d/s_n be large. Detection of rank deficiency by checking the size of diagonal entries of R will fail only as indicated in the following theorem.

THEOREM 2.8. *For A, d, and* w_k *as defined earlier, and for* $1 \le B \le C = s_1/s_n$, *if* $d/s_n \ge B$, *then for any* $k \ge 1$

$$(2.9) \qquad w_{n-k+1} \le \frac{(C/B)^{2k} - 1}{C^2 - 1}.$$

Proof. If $B \le d/s_n$, then $B \le r_{kk}(A)/s_n$, $k = 1, \cdots, n$. Therefore by Theorem 2.6, $B^2 \le C^2[w_{k+1} + (1 - w_{k+1})/C^2]/[w_k + (1 - w_k)/C^2]$ or $w_k \le (C/B)^2 w_{k+1} +$

$[(C/B)^2 - 1]/(C^2 - 1)$. Since $w_{n+1} = 0$, the inequality in the theorem follows by induction. We might note here that the inequality (2.9) is trivial for $k > \ln C/\ln(C/B)$ since, for such k, $[(C/B)^{2k} - 1]/(C^2 - 1) \geq 1$ and by definition $w_k \leq 1$. □

The conclusion of the theorem may become clearer if we consider $B = C^{1-1/p}$, $1 \leq p \leq n$. The theorem asserts for $1 \leq k \leq p$ that

$$d/s_n \geq C^{1-1/p} \Rightarrow v_{n-k+1,n}^2 \leq w_{n-k+1} \leq \frac{(C/B)^{2k} - 1}{C^2 - 1} \leq C^{2(k/p-1)} \tag{2.10}$$

where, for $1 \leq k \leq p$, the last inequality follows by algebra. Qualitatively, inequalities (2.10) state that if C is large and $d/s_n \geq C^{1-1/p}$, then the last $p-1$ components of $\mathbf{v}_n$ must be small.

The condition (2.9) is not a sufficient condition that $d/s_n \geq B$, as is illustrated by

$$A = \begin{pmatrix} 0 & 1 \\ \epsilon & 0 \end{pmatrix}, \quad \epsilon < 1.$$

In this case $\mathbf{v}_n = (1,0)^T$, $d = \epsilon$, $s_n = \epsilon$, and $C = 1/\epsilon$. If we choose $B = 1/\sqrt{\epsilon}$, then $w_2 = 0 \leq [(C/B)^2 - 1]/[C^2 - 1]$ and $w_1 = 1 = [(C/B)^4 - 1]/[C^2 - 1]$ satisfy (2.9) but, of course, $d/s_n = 1 \not\geq B$. Therefore, we wish to consider sufficient conditions that d/s_n is large. Such conditions are important, since they can be viewed as describing conditions for which it is guaranteed that detection of rank deficiency by examining d will be difficult or impossible.

THEOREM 2.11. *Let A, d, and w_k be as defined earlier and let $G = s_{n-1}/s_n$. For some k, $2 \leq k \leq n$, suppose $1 \leq B \leq G^{1-1/(n-k+2)}$ and let $\beta^2 = [(G/B)^2 - 1]/(G^2 - 1)$. If*

$$\begin{gathered} w_n \leq \beta^2, w_j \leq (G/B)^2 w_{j+1} + \beta^2, \quad j = k, \cdots, n-1, \\ \text{and} \quad w_k \geq (B/G)^2(1 - \beta^2) \end{gathered} \tag{2.12}$$

then $d/s_n \geq B$.

Proof. The first inequality in (2.12) is equivalent to $B^2 \leq 1/[w_n + (1 - w_n)/G^2]$ and therefore by Theorem 2.6, $B^2 \leq r_{nn}^2/s_n^2$. The second set of inequalities in (2.12) is equivalent to $B^2 \leq G^2[w_{j+1} + (1 - w_{j+1})/G^2]/[w_j + (1 - w_j)/G^2]$, $\quad j = k, \cdots, n-1$ and therefore by Theorem 2.6, $B^2 \leq r_{jj}^2/s_n^2$, $\quad j = k, \cdots, n-1$. The third inequality in (2.12) is equivalent to $B^2 \leq G^2[w_k + (1 - w_k)/G^2]$. Therefore by Corollary 2.7 $B^2 \leq r_{jj}^2/s_n^2$, $\quad j = 1, \cdots, k-1$.

The requirement that $B \leq G^{1-1/(n-k+2)}$ is included, since (2.12) is inconsistent otherwise. □

The meaning of this theorem becomes clearer if we look at the special case where $k = n$. Then (2.12) implies for $1 \leq B \leq \sqrt{G}$ that

$$(B/G)(1 - \beta^2)^{1/2} \leq |v_{nn}| \leq \beta \Rightarrow d/s_n \geq B. \tag{2.13}$$

In the case that $1 << B << \sqrt{G}$, then $\beta \cong 1/B$ and $1 - \beta^2 \cong 1$ and so qualitatively (2.13) means for matrices with a large "gap" G that if $|v_{nn}|$ is small, but not too small, then it follows that d will be much larger than s_n . Therefore in this case detection of rank deficiency by examination of d will be difficult or impossible.

Finally, in this section we wish to present a corollary to Theorem 2.11 that is simpler although somewhat weaker.

COROLLARY 2.14. *Let A and d be as defined earlier and $G = s_{n-1}/s_n$. For some k, $2 \le k \le n$, suppose $1 \le B \le G^{1-1/(n-k+2)}$ and let $\beta^2 = [(G/B)^2 - 1]/(G^2 - 1)$. If*

$$(2.15)\quad |v_{nn}| \le \beta, \quad |v_{jn}| \le (G/B)|v_{j+1,n}|, \quad j = k, \cdots, n-1, \quad \text{and} \quad |v_{kn}| \ge B/G$$

then $d/s_n \ge B$.

Proof. Inequalities 2.15 imply that $(B/G)^2 \le v_{kn}^2$, $(B/G)^2 v_{jn}^2 \le v_{j+1,n}^2$, $j = k, \cdots, n-1$, and $(B/G)^2 v_{nn}^2 \le (\beta B/G)^2$. Adding these inequalities and noting that $w_k = v_{kn}^2 + \cdots + v_{nn}^2$, we obtain $(B/G)^2(1 + w_k) \le w_k + (\beta B/G)^2$ or $(B/G)^2(1-\beta^2) \le (1 - B^2/G^2)w_k \le w_k$. This is the last inequality in (2.12). For $k \le j \le n$, adding $[(B/G)v_{in}]^2 \le v_{i+1,n}^2$, $i = j, \cdots, n-1$, and $[(B/G)v_{nn}]^2 \le (\beta B/G)^2$, we obtain $(B/G)^2 w_j \le w_{j+1} + (\beta B/G)^2$ or $w_j \le (G/B)^2 w_{j+1} + \beta^2$, $j = k, \cdots, n$. Recalling that $w_{n+1} = 0$, these are the remaining inequalities in (2.11). Therefore by Theorem 2.11 $d/s_n \ge B$. □

To help clarify the meaning of this corollary, let us assume that $1 << B \le G$. Then $\beta \cong 1/B$ and (2.15) asserts that if $|v_{nn}|$ is small, if the components $|v_{jn}|, j = n-1, n-2, \cdots, k$ are bounded by an exponential growth with factor $G/B \ge 1$, and if $|v_{kn}|$ is not too small, then d/s_n will be large. Smaller k values imply results for larger values of B.

3. The probability of large diagonal entries.

There are two major difficulties in estimating the probability of rare events related to a matrix decomposition such as the QR decomposition: (1) if A is an $m \times n$ dense matrix, then the cost of determining the factorization is proportional to mn^2 and for moderate or large n this cost can be nontrivial; and (2) if one attempts to determine such a probability by direct simulation the sample size must be very large if the event is indeed rare. This combination can make direct simulation too costly to be practical: the calculations for each sample point are nontrivial and the sample size must be huge. This, in part, is the reason that unresolved problems relating to rare events in matrix decomposition remain.

With the aid of the results for §2 we overcome both the above difficulties for our problem. For the first difficulty we can use Theorem 2.6 and Corollary 2.7 to calculate upper and lower bounds on d. As we will see, these bounds can usually be generated in $O(1)$ not $O(mn^2)$ operations and they lead to upper and lower bounds on the desired probability. To overcome the second difficulty, we will generate only sample matrices A that correspond to necessary conditions similar to (2.9). This leads to a dramatic reduction in the sample size needed for accurate calculation of probabilities.

To develop this scheme, we will first describe the class of random matrices A to be examined. We then discuss efficient ways to generate these matrices and to test that $d/s_n \ge B$. Finally, we present the results of our computer simulations.

In order to have some sense of how often a certain algorithm will fail to work, one could test the algorithm on matrices arising from "real world" problems or on analytically generated matrices. In this paper we have chosen the latter approach, since it is difficult to collect and test large sets of real world matrices (see the above comments) and since there exists a natural class of analytically generated matrices that has been used elsewhere. Our class of random matrices will be a generalization of that described by Stewart [S2]. In order to describe this class, we first introduce a class of random orthogonal matrices as follows: let X be an $m \times n$ matrix whose entries x_{ij} are chosen so that x_{ij}^2 belongs to a gamma distribution with parameters a and b; $a, b > 0$ (and so has a density function $x^{a-1}e^{x/b}/[b^a\Gamma(a)]$). The sign of x_{ij}

should be chosen positive or negative with equal probability. Let $X = \hat{Q}\hat{R}$ so that $\hat{Q}$ is the Gram–Schmidt orthonormalization of the columns of X. Let P be the permutation matrix with ones down the skew diagonal (i.e., $p_{i,n-i+1} = 1$) and, finally, define the random orthogonal $n \times n$ matrix $V = \hat{Q}P$. We now define our class of random $m \times n$ general matrices by $A = UDV^T$, where D is an $m \times n$ diagonal matrix with specified diagonal entries $s_1 \geq s_2 \geq \cdots \geq s_n$ and U is an arbitrary $m \times m$ orthogonal matrix.

In the case that $a = 1/2$ the $n \times n$ orthogonal matrices V follows the distribution corresponding to Haar measure [Hal]. Thus in this case V is from the natural or uniform distribution of orthogonal matrices in that the measure of a set of such matrices is invariant under multiplication by an orthogonal matrix [Hal], [S2]. This is the class of matrices discussed in [S2]. A very similar class was used in [Hi2] and matrices V from the Haar distribution have been used by [BM], [Hei], and [JW]. Also in this case the permutation matrix P is irrelevant, since the distribution is invariant under orthogonal transformations. Although for $a = 1/2$ we could omit P without loss of generality, as we will see for $a \neq 1/2$ we need P for technical reasons. We consider cases where $a \neq 1/2$ for generality. For example, for $a < 1/2$ the probability that certain elements of V are small is substantially increased. Thus for $n = 50$ our results will show that for a = .5, the median of $|v_{nn}|$ is .096 and $P(|v_{nn}| \leq .0018) = .01$ where as for $a = .1$ the median of $|v_{nn}|$ is .011 and $P(|v_{nn}| \leq 3.7 \times 10^{-11}) = .01$. We should also note that U above is arbitrary, since its selection does not affect our results. We select D deterministically in order to control the spectrum of A.

Our first theorem of this section follows.

THEOREM 3.1. *If $x_{i1}, i = 1, \cdots, n$, are the elements of the first column of the matrix X described earlier and $z_k = x_{k1}^2 / \sum_{i=1}^{k} x_{i1}^2, \quad k = 2, \cdots, n$, then $z_k, k = 2, \cdots, n$ are mutually independent and z_k has a beta distribution with parameters a and $(k-1)a$.*

Proof. Since x_{i1}^2 follows a gamma distribution with parameters a and b, then by [R, Thm. 4, p. 208], $\sum_{i=1}^{k-1} x_{i1}^2$ is a gamma distribution with parameters $(k-1)a$ and b. By [R, Thm. 15, p. 214] we may conclude that z_k is a beta distribution with density function $f(z) = \Gamma(ka)z^{a-1}(1-z)^{(k-1)a-1}/[\Gamma(ka-a)\Gamma(a)]$. By standard techniques ([HoC, pp. 129-134]) it can then be shown that the joint probability density function of $z_2, z_3, \cdots, z_n$ is the product of the individual density functions and so $z_k, \quad k = 2, 3, \cdots, n$ are independent. □

COROLLARY 3.2. *Let z_k be as above and let $w_k = \sum_{i=k}^{n} v_{in}^2$ where V is the random orthogonal matrix described earlier. Then for each k, $2 \leq k \leq n$, z_k is independent of $w_j, j = k+1, \cdots, n$, and if $w_{n+1} = 0$, then $w_k = w_{k+1} + z_k(1 - w_{k+1}), \quad k = 2, \cdots, n$.*

Proof. Since $V = \hat{Q}P$, where P is the permutation matrix with ones down the skew diagonal, the last column of V is the first column of $\hat{Q}$. However, since $X = \hat{Q}R$, the first column of $\hat{Q}$ is the first column of X after normalization. Therefore $v_{kn}^2 = x_{k1}^2/\sum_{i=1}^{n} x_{i1}^2$ and $w_k = \sum_{i=k}^{n} x_{i1}^2/\sum_{i=1}^{n} x_{i1}^2, \quad k = 1, \cdots, n$. Since $z_k = x_{k1}^2/\sum_{i=1}^{k} x_{i1}^2$, then $w_k = w_{k+1} + z_k(1 - w_{k+1})$ follows by algebra. Also it follows by algebra that, for $j = 2, \cdots, n, \quad w_j = 1 - \prod_{i=j}^{n}(1 - z_i)$. Since $z_j, j = k, \cdots, n$ are mutually independent, it then follows that z_k is independent of

$$w_j, j = k+1, \cdots, n. \qquad \square$$

Our first theorem concerning the probability of large diagonal entries in a QR factorization is given in Theorem 3.3.

THEOREM 3.3. *For random $m \times n$ matrices A, generated as above, let $C = s_1/s_n$, let B satisfy $1 \leq B \leq C$, and let l be equal to the largest integer less than*

$\min\{\ln B/(\ln C - \ln B), n-1\}$. *If random variables z_k are as in Theorem* 3.1, *then*

$$(3.4)\qquad P(d/s_n \geq B) \leq \prod_{k=0}^{l} P\{z_{n-k} \leq [(C/B)^2 - 1](C/B)^{2k}/[C^2 - (C/B)^{2k}]\}.$$

Proof. By Corollary 3.2 $z_k = (w_k - w_{k+1})/(1 - w_{k+1})$, $k = 2, \cdots, n$ and, as shown in the proof of Theorem 2.8, if $d/s_n \geq B$, then $w_k \leq (C/B)^2 w_{k+1} + [(C/B)^2 - 1]/C^2 - 1), k = 1, \cdots, n$. These results imply that $z_k \leq [(C^2/B^2 - 1)/(C^2 - 1)][1 + (C^2 - 1)w_{k+1}]/(1 - w_{k+1})$, $k = 2, \cdots, n$. However, by Theorem 2.8, $d/s_n \geq B$ implies that $w_{n-k+1} \leq [(C/B)^{2k} - 1]/(C^2 - 1)$. Since $[1 + (C^2 - 1)w_{k+1}]/(1 - w_{k+1})$ is an increasing function of w_{k+1} for $0 < w_{k+1} < 1$, and since the bound on w_{n-k+1} in (2.9) is less than 1 for $k < \ln C/\ln(C/B)$, it follows after some algebra that if $d/s_n \geq B$, then

$$(3.5)\qquad z_{n-k} \leq [(C/B)^2 - 1](C/B)^{2k}/[C^2 - (C/B)^{2k}], \qquad 0 \leq k < \ln C/\ln(C/B).$$

The theorem now follows, since by Theorem 3.1 the z_k, $k = 2, \cdots, n$, are independent. Note that the restriction $l < n - 1$ is in the theorem statement, since $z_1 \equiv 1$ (see Theorem 3.1) is not a random variable. □

Since the random variables z_k, $k = 2, 3, \cdots, n$, follow beta distributions, it is elementary to calculate the probabilities in (3.4) via an appropriate IMSL routine or CACM Algorithm 179. Equation (3.4) provides the following upper bounds on $P(d/s_n \geq B)$ for the case that $a = 1/2$ (so the orthogonal matrices are from the Haar distribution).

TABLE 3.6

Upper bounds on $P(d/s_n \geq B)$ for $m \times n$ matrices with condition number C. The random matrices A have parameter $a = 1/2$.

n	10	10	50	50	1000	1000
$B \setminus C$	10^4	10^6	10^4	10^6	10^4	10^6
10	.230	.230	.515	.515	.998	.998
100	2.3×10^{-2}	2.3×10^{-2}	5.6×10^{-2}	5.6×10^{-2}	.248	.248
1000	1.0×10^{-5}	2.3×10^{-3}	1.5×10^{-4}	5.6×10^{-3}	6.2×10^{-3}	2.5×10^{-2}
10^4	-	5.1×10^{-6}	-	3.1×10^{-5}	-	6.2×10^{-4}
10^5	-	3.2×10^{-14}	-	4.3×10^{-12}	-	3.9×10^{-9}

From this table it is clear that the probability that d/s_n is close to the condition number C is very small. (For example, when $n = 10$ and $C = 10^6$, then $P(d/s_n \geq 10^5) \leq 3.20 \times 10^{-14}$)! However, we would like estimates of both upper and lower bounds on $P(d/s_n \geq B)$ for B substantially smaller than C, and we would like to get asymptotic estimates of $P(d/s_n \geq B)$. The latter is provided by the following result.

THEOREM 3.7. *For random matrices A as above with gap $G = s_{n-1}/s_n$ then*

$$\lim_{G\to\infty} P(d/s_n \geq B) = P(z_n \leq 1/B^2).$$

Proof. Let $\beta^2 = (G^2/B^2 - 1)/(G^2 - 1)$ as in Theorem 2.11. Then by (2.12), and since $z_n = w_n = v_{nn}^2$, it follows that if $B \leq \sqrt{G}$, then $P[(B^2/G^2)(1-\beta^2) \leq z_n \leq \beta^2] \leq P(d/s_n \geq B)$. Due to (2.9), and since for $1 \leq B \leq C$, $(C^2/B^2 - 1)/(C^2 - 1) \leq 1/B^2$, it follows that $P(d/s_n \geq B) \leq P(z_n \leq (C^2/B^2 - 1)/(C^2 - 1)) \leq P(z_n \leq 1/B^2)$. By algebra it follows that as $G \to \infty$, $(B^2/G^2)(1-\beta^2) \to 0$ and $\beta^2 \to 1/B^2$. Since z_n has a continuous density function the result follows. $\square$

Since z_n is distributed as a beta distribution with parameters a and $(n-1)a$ (Theorem 3.1), it follows easily that for sufficiently large B, $P(z_n \le 1/B^2) \cong \Gamma(an)(1/B^2)/[\Gamma(a+1)\Gamma(na-a)]$. Therefore by Theorem 3.7 if G is also large, then

$$P(d/s_n \ge B) \cong \Gamma(an)(1/B^2)^a/[\Gamma(a+1)\Gamma(na-a)]. \tag{3.8}$$

If we make the natural selection $a = 1/2$ (see our earlier discussion) and assume that $1 << n$ ($10 \le n$, say), then (3.8) reduces to

$$P(d/s_n \ge B) \cong \sqrt{2n/\pi}(1/B). \tag{3.9}$$

For example, for $n = 50$, $B = 100$, and $G >> 10^4$ it follows that $P(d/s_n \ge B) \cong 5.6$ percent.

In order to obtain estimates of upper bounds on $P(d/s_n \ge B)$ that are sharper than the results of Table 3.6, and to obtain estimates of lower bounds on $P(d/s_n \ge B)$, we carried out a simulation as follows. For any spectrum $s_1 \ge s_2 \ge \cdots \ge s_n$ and the corresponding random matrix $A = UDV^T$, as above, consider the diagonal matrices $D_* = \operatorname{diag}(s_{n-1}, s_{n-1}, \cdots, s_{n-1}, s_n)$ and $D^* = \operatorname{diag}(s_1, s_1, \cdots, s_1, s_n)$, the random matrices $A_* = UD_*V^T$ and $A^* = UD^*V^T$ and the minimum diagonal entries d_* and d^*, respectively, in QR factorizations of A_* and A^*. By Lemma 2.3 $d_* \le d \le d^*$ and so

$$P(d_*/s_n \ge B) \le P(d/s_n \ge B) \le P(d^*/s_n \ge B). \tag{3.10}$$

However, $P(d_*/s_n \ge B)$ and $P(d^*/s_n \ge B)$ can be estimated very efficiently in a similar manner.

We will describe the technique for calculating these probabilities in terms of $P(d^*/s_n \ge B)$. Lemma 2.4, Corollary 2.7, Theorem 3.1, and Corollary 3.2 imply that the following algorithm will generate a sample from the distribution of d^*'s:

(1) Let $k = n$.

(2) Generate z_k from a beta distribution with parameters a and $(k-1)a$.

(3) Calculate $w_k = w_{k+1} + z_k(1 - w_{k+1})$ and r_{kk} using (2.5) and let $d_k = \min_{k \le i \le n} r_{ii}$.

(4) Test if $d_k \le \min_{1 \le j \le k} r_{jj}$ by using Corollary 2.7. If so, accept $d^* = d_k$ and stop, else let $k = k - 1$ and go to step 2.

The calculations involved in steps 2, 3, and 4 can each be done in a few operations. Furthermore, in our experiments typically the test in step 4 was passed for k close to n. In most of our runs $n - k$ was less than 5. Therefore a sample value of d^* can be generated in a number of operations largely independent of n.

Using this algorithm, we carried out a simulation to estimate $P(d^*/s_n \ge B)$. In order to reduce the number of samples required for a given B, the only z_k's generated were those corresponding to the necessary conditions (3.5). To see how this is done, let S_2 be the set of random matrices A^* that come from z_k's restricted by (3.5) and let S_1 be all random matrices A^* with $d^*/s_n \ge B$. Then since $S_1 \subseteq S_2$ (see Theorem 3.3 and its proof), it follows that $P(d^*/s_n \ge B) = P(S_1) = P(S_1|S_2)P(S_2)$, where "|" is used to indicate conditional probability. However, $P(S_2)$ is equal to the right hand side of (3.4), and $P(S_1|S_2)$ was estimated via a simulation by calculating the relative frequency of success (i.e., $d^*/s_n \ge B$) of samples chosen from S_2. Such a procedure reduces the number of samples required to accurately estimate $P(S_1)$ by factors equal to the probabilities listed in Table 3.6, a reduction of up to a factor of

10^{-14}! We should add that the generation of random samples from a beta distribution with the restriction (3.5) can be done with a straightforward, efficient modification of the rejection method for generation of random numbers.

The number of samples in each simulation was chosen large enough so that an estimate σ_p of the standard deviation of the relative frequency of success was less than some preset tolerance. (See pp. 201-202 of [HoC] for the usual estimate of such a standard deviation.) In all our runs the sample size was sufficiently large so that the estimated value of $\sigma_p/P(d^*/s_n \geq B)$ was less than 5 percent. Thus (see [HoC]) our estimated values of $P(d^*/s_n \geq B)$ are correct to within ± 20 percent with approximately a .99994 confidence level.

The above algorithm is also directly applicable to calculate $P(d_*/s_n \geq B)$. The sole change is that the gap and the condition number of A^* are both s_1/s_n, whereas the gap and the condition number of A_* are s_{n-1}/s_n.

We carried out the simulations described above and the results are contained in Table 3.11. If, as earlier, for $A = UDV^T$ we let $G = G(A) = s_{n-1}/s_n$ and $C = C(A) = s_1/s_n$, then entries in the table below provide estimates of upper bounds on $P(d/s_n \geq B)$ if A has the indicated condition number and lower bounds if A has the indicated gap. For example, in the following table if a 50×50 matrix has a gap of 10^4 and condition number of 10^6 and if the estimated probabilities are exact, then $.049 \leq P(d/s_n \geq B) \leq .055$ for $B = 100$. Since the uncertainties in the estimated probabilities are less than 20 percent (with a .99994 confidence level), we may conclude that $.039 \leq P(d/s_n \geq B) \leq .066$. Table 3.11 is for random matrices generated with parameter $a = 1/2$ so that the orthogonal matrices used to generate A are from the natural or Haar distribution.

TABLE 3.11

Estimates of lower bounds on $P(d/s_n \geq B)$ for $m \times n$ matrices with gap G and upper bounds on $P(d/s_n \geq B)$ for matrices with a condition number C. The random matrices A have parameter $a = 1/2$.

n	10	10	10	50	50	50
$B \setminus G$ or C	10^4	10^6	10^{10}	10^4	10^6	10^{10}
10	.23	.23	.23	.51	.51	.51
100	.017	.023	.023	.049	.055	.055
1000	6.7×10^{-7}	1.7×10^{-3}	2×10^{-3}	2×10^{-5}	4.9×10^{-3}	5.5×10^{-3}
10^4	-	1.3×10^{-6}	2×10^{-4}	-	1.2×10^{-5}	5.5×10^{-4}
10^5	-	5.3×10^{-16}	2×10^{-5}	-	2.1×10^{-14}	4.9×10^{-5}

n	1000	1000	1000
$B \setminus G$ or C	10^4	10^6	10^{10}
10	.998	.998	.998
100	.24	.25	.25
1000	1.8×10^{-3}	2.5×10^{-2}	2.5×10^{-2}
10^4	-	3.2×10^{-4}	2.5×10^{-3}
10^5	-	6.1×10^{-11}	2.5×10^{-4}

These results clearly support our earlier comment that d/s_n is large with a significant probability for a natural class of random matrices. For example, for 50×50 matrices with a gap of 10^4 an estimate for the lower bound on the probability that $d/s_n \geq 100$ is .049. In such cases it would be very difficult to detect rank deficiency using d.

In this section we would also like to present some results for a different selection of the parameter a defining the random orthogonal matrix used to construct A. For $a = .1$ we obtain the following results.

TABLE 3.12

Estimates of lower bounds on $P(d/s_n \geq B)$ for $m \times n$ matrices with gap G and upper bounds for matrices with condition number C. The random matrices A have parameter $a = .1$.

n	50	50	50
$B \setminus G$ or C	10^4	10^6	10^{10}
10	.71	.75	.76
100	.25	.41	.48
1000	3.8×10^{-3}	.15	.29
10^4	-	1.5×10^{-2}	.16
10^5	-	1.2×10^{-6}	6.9×10^{-2}
10^7	-	-	7.5×10^{-4}

Comparing Table 3.11 and Table 3.12, it is clear that reducing a can substantially increase the probability that d is much larger than s_n. In more general terms this shows that if the last components of nth singular vector of A are for some reason likely to be small, then it becomes more likely that rank detection by examining diagonal entries of R in $A = QR$ factors will fail. Conversely, if these last components of $\mathbf{v}_n$ are not likely to be small, then it is more probable the $d \cong s_n$.

Finally, in this section we would like to present some results for matrices with a cluster of small singular values, since the sufficient conditions in §2 and the lower probability bounds in this section produce trivial results if s_{n-1}/s_n is not large. In the interests of space, we will summarize our results but not present the details of our development for matrices with several small singular values. It can be shown that for a matrix A with a large condition number C and with $l \geq 1$ small singular values, if d/s_n is sufficiently large, then the last few components of the last l singular vectors of A must be small. Furthermore, one can show that for a matrix with a cluster of l small singular values (and whose other singular values are not small) if the the last l components of the last l singular vectors are small and if the smallest singular value of the $l \times l$ matrix formed by these components is not too small, then it follows that d/s_n is large.

The above conditions are more stringent for $l > 1$ than for $l = 1$ and therefore for $l > 1$ it would appear less likely that d/s_n would be large. This can be verified by simulations for $l > 1$ similar to those detailed earlier for $l = 1$. Some of our results for $l = 2$ are illustrated in Table 3.13. In the table if the indicated singular value ratio (SVR) is interpreted as s_{n-2}/s_n, then the table provides lower bounds on $P(d/s_n \geq B)$ and if SVR is interpreted as s_1/s_{n-1}, then upper bounds are provided.

For matrices with a single small singular value ($l = 1$) the above calculations were done on a Sequent Balance 8000 computer. For these matrices the calculation of each probability in the tables typically required a few seconds of CPU time. For matrices with several small singular values the computations were more difficult, since the necessary conditions used were not as tight and since the technique used to generate samples for $l > 1$ required $O(n)$, not $O(1)$, operations per sample. The calculations for these matrices were done on a CRAY-XMP/48 computer and the CPU times required ranged up to four minutes to calculate some of the probabilities. Also we should note here that for smaller matrices ($n \leq 10$, say) and probabilities that are not too small ($P > .01$, say) it is possible to compare the calculations outlined above with direct simulation based on forming QR factorizations of random matrices A. Such comparisons agreed in all cases tested.

TABLE 3.13

Estimates of lower bounds on $P(d/s_n \geq B)$ for $m \times n$ matrices when SVR *is interpreted as s_{n-2}/s_n and upper bounds when* SVR *is interpreted as s_1/s_{n-1}. The random matrices A have parameter $a = 1/2$ and $n = 50$.*

$B \setminus$ SVR	10^4	10^6	10^{10}
10	.11	.11	.11
100	3.2×10^{-5}	1.1×10^{-4}	1.3×10^{-4}

4. Conclusions and extensions. For any matrix A we have provided relatively simple necessary conditions and separate sufficient conditions that the smallest diagonal entry d in a QR factorization of A is much larger than the smallest diagonal entry of A. From these conditions it then followed for certain natural random matrices with an isolated small singular value that the probability that d is much bigger than s_n is not insignificant. If the distribution of random matrices is chosen so that the probability of having small entries in the nth singular vector of A is increased, then the probability that d/s_n is large also increases. Finally, we showed that if the matrices A have several small singular values, then the probability that d/s_n is large substantially decreases in comparison to matrices with a single small singular value.

The results summarized above suggest that in practice one should be cautious in using d for the determination of rank deficiency. Of course, our results in §3 are based on a study of a natural class of analytically generated matrices. Additional work with matrices from "real world" problems would be a useful extension of our work. We should note that some experimentation with real world matrices has been briefly described elsewhere. Heath [He, p. 228] mentions that an algorithm for testing rank deficiency based on the size of d worked well in practice, whereas Golub and Wilkinson [GW, p. 593] reported that "almost invariably some of the R were such that they had no small r_{ii}" when using QR factorizations to check for rank deficiency as part of a calculation of Jordan canonical form. Our work in §3 with analytically generated matrices is of independent interest. In addition, we should note that the necessary and sufficient conditions of Chapter 2 apply to any matrix whether it is analytically generated or from the "real world."

Some of the ideas and tools introduced to determine the probabilities included using necessary conditions to restrict the sample size in a simulation and the use of an efficient technique to generate samples. These tools allowed calculation of exceedingly rare events (probabilities as small as 10^{-16}) for matrices that were quite large (up to 1000×1000). We hope that some of the ideas introduced will be helpful in solving other open problems relating to rare events in matrix decomposition.

Acknowledgments. The author is grateful for the thorough examination and suggestions of the referee.

REFERENCES

[AG] E. ALLGOWER AND K. GEORG, *Simplicial and continuation methods for approximating fixed points and solutions to systems of equations*, SIAM Rev., 22, pp. 28–85.

[AH] H. C. ANDREWS AND B. R. HUNT, *Digital image restoration*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

[B] Å. BJÖRK, *A general updating algorithm for constrained linear least squares problems*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 394–402.

[BM] R. BENDEL AND M. MICKEY, *Population correlation matrices for sampling experiments*, Commun. Statist., B7-2 (1978), pp. 163–182.

[C1] T. CHAN, *On the existence and computation of LU- factorizations with small pivots*, Math. Comp., 42 (1984), pp. 535–547.

[C2] ———, *Rank revealing QR factorizations*, Linear Algebra Appl., 88/89 (1987), pp. 67–82.

[CMSW] A. CLINE, C. MOLER, G. STEWART, AND J. WILKINSON, *An estimate for the condition number of a matrix*, SIAM J. Numer. Anal., 16 (1979), pp. 368–375.

[CR] A. CLINE AND R. REW, *A set of counterexamples to three condition number estimations*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 602–611.

[DBMS] J. DONGARRA, J. BUNCH, C. MOLER, AND G. STEWART, *Linpack User's Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1979.

[EM] U. ECKHARDT AND K. MIKA, *Numerical treatment of incorrectly posed problems— A case study*, in Numerical Treatment of Integral Equations, J. Albrecht and L. Collatz, eds., Workshop on numerical Treatment of Integral Equations, Oberwolfach, Germany. November 18-24, 1979, Birkhauser Verlag, Basel, Switzerland, 1980, pp. 92-101.

[E] L. ELDEN, *Algorithms for regularization of ill- conditioned least squares problems*, BIT, 17 (1977), pp. 134–145.

[F] L. FOSTER, *Rank and null space calculation using matrix decomposition without column interchanges*, Linear Algebra Appl., 74 (1986), pp. 47-71.

[GN] A. GEORGE AND E. NG, SPARSPAK*: Waterloo Sparse Matrix Package User's Guide for* SPARSPAK-B, Res. Report CS-84-37, Univ. of Waterloo, Waterloo, Ontario, Canada, 1984.

[GKS] G. GOLUB, V. KLEMA, AND G. STEWART, *Rank deficiency and least squares problems*, Tech. Report STAN-CS-76-559, Stanford University, Stanford, CA, 1976.

[GVL] G. GOLUB AND C. VAN LOAN, *Matrix Computations*, Johns Hopkins Press, Baltimore, 1983.

[GW] G. GOLUB AND J. WILKINSON, *Ill-conditioned eigensystems and the computation of Jordan canonical form*, SIAM Rev., 18 (1976), pp. 578-619.

[Hal] P. HALMOS, *Measure Theory*, Van Nostrand, Princeton, NJ, 1950.

[HaC] P. C. HANSEN AND S. CHRISTIANSEN, *An* SVD *analysis of linear algebraic equations derived from first kind integral equations*, J. Comput. Appl. Math., 12/13 (1985), pp. 341–357.

[Han] R. J. HANSON, *A numerical method for solving Fredholm integral equations of the first kind using singular values*, SIAM J. Numer. Anal., 8 (1971), pp. 616–622.

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

[Hei] R. HEIBERGER, *Generation of random orthogonal matrices*, Appl. Statist., 27-2 (1978), pp. 199- 206.

[Hi1] N. HIGHAM, *A survey of condition number estimation for triangular matrices*, SIAM Rev., (1987), pp. 575–596.

[Hi2] ———, *Computing the polar decomposition with applications*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1160–1174.

[HoC] R. HOGG AND A. CRAIG, *Introduction to Mathematical Statistics*, Macmillan, New York, 1969.

[JW] D. JOHNSON AND W. WELCH, *The generation of pseudo-random correlation matrices*, J. Statist. Comput. Simulation, 11 (1980), pp. 55–59.

[Ka] W. KAHAN, *Numerical linear algebra*, Canadian Math. Bull., 9 (1966), pp. 757–801.

[Ke] H. B. KELLER, *Numerical solutions of bifurcation and nonlinear eigenvalue problems*, in Applications of Bifurcation Theory, P. Rabinowitz, ed., Academic Press, New York, 1977, pp. 359–384.

[LH] C. LAWSON AND R. HANSON, *Solving least squares problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[LN] A. K. LOUIS AND F. NATTERER, *Mathematical problems of computerized tomography*, Proc. IEEE, 71 (1983), pp. 379–389.

[MR] R. G. MELHELM AND W. C. RHEINBOLDT, *A comparison of methods for determining turning points of nonlinear equations*, Computing, 29 (1982), pp. 201–226.

[ML] B. C. MOORE AND A. J. LAUB, *Computation of supremal (A, B)-invariant and controllability subspaces*, IEEE Trans. Automat. Contr., AC-23 (1978), pp. 783-792.

[N] F. NATTERER, *Numerical inversion of the radon transform*, Numer. Math., 30 (1978), pp. 81-91.

[P] D. L. PHILLIPS, *A technique for the numerical solution of certain integral equations of the first kind*, J. Assoc. Comput. Mach., 9 (1962), pp. 84-97.

[R] V. ROHATGI, *An introduction to probability theory and mathematical statistics*, John Wiley, New York, 1976.

[S1] G. STEWART, *Computable error bounds for aggregated Markov chains*, Tech. Report 901, Univ. Maryland Computer Science Center, College Park, MD, 1980.

[S2] ———, *The efficient generation of random orthogonal matrices with an application to condition estimators*, SIAM J. Numer. Anal., 17 (1980), pp. 403–409.

[TA] A. TIKHONOV AND V. ARSENIN, *Solution of ill-posed problems*, V.H. Winston and Sons, 1977.

[T] L. TREFETHEN, *Three mysteries of Gaussian elimination*, ACM Signum Newsletter, 20 (1985), pp. 2–5.

[TK] D. W. TUFTS AND R. KUMARESAN, *Singular value decomposition and improved frequency estimation using linear prediction*, IEEE Trans. Acoust. Speech, Signal Processing, ASSP-30 (1982), pp. 671–675.

[VD] P. M. VAN DOOREN, *The generalized eigenstructure problem in linear system theory*, IEEE Trans. Automat. Contr., AC-26 (1981), pp. 111-129.

[V] J. M. VARAH, *On the numerical solution of ill-conditioned linear systems with applications to ill-posed problems*, SIAM J. Numer. Anal., 10 (1973), pp. 257–267.

[ZWS] Z. ZLATEV, J. WASNIEWSKI, AND K. SCHAUINBURG, *Condition number estimators in a sparse matrix software*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1175–1189.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 3, pp. 545-562, May 1990

INCOMPLETE BLOCK CYCLIC REDUCTION AS A PRECONDITIONER FOR POLYNOMIAL ITERATIVE METHODS*

I. K. ABU-SHUMAYS†

Abstract. Preconditionings based on incomplete block odd-even cyclic reduction for the Chebyshev and conjugate gradient polynomial iterative methods have been shown to be effective for accelerating the convergence of the solution of linear systems that arise from discrete approximations of elliptic partial differential equations. The main contribution of this paper is to extend these methods to the important case of red/black partitioning of equations and unknowns and to demonstrate that a significant reduction in computations can be realized by applying preconditioned polynomial methods to the "reduced systems" obtained from the original systems by eliminating approximately half of the unknowns.

This paper also compares the performance of several preconditioned polynomial methods for solving a two-dimensional elliptic equation on the CYBER 205 vector computer. The most effective methods tested are shown to be (i) the new reduced system conjugate gradient method with preconditioner based on incomplete block odd-even cyclic reduction, and (ii) the classical reduced system conjugate gradient method with preconditioner D_B, the tridiagonal matrix associated with the black unknowns. Of these methods, the former is better in most cases, especially for some slowly converging problems.

Key words. conjugate gradient method, preconditioning, block tridiagonal systems, red/black ordering, odd-even cyclic reduction, incomplete factorization, Chebyshev method

AMS(MOS) subject classifications. 65F10, 65B99, 65N20

1. Introduction.

Consider the block tridiagonal linear system

$$(1)\qquad Ax=\begin{bmatrix} A_1 & B_1 & & & \\ B_1^T & A_2 & B_2 & & \\ & B_2^T & \ddots & \ddots & \\ & & \ddots & \ddots & B_{n-1} \\ & & & B_{n-1}^T & A_n \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{bmatrix}=\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_n \end{bmatrix}=b$$

where the matrix A is symmetric positive definite. The transformation

$$(2)\qquad x_{RB}=Ux,\qquad b_{RB}=Ub,\qquad A_{RB}=UAU^T$$

where U is the permutation matrix that separates the odd-numbered (red) and even-numbered (black) unknowns, renders (1) into the red/black form

$$(3)\qquad A_{RB}x_{RB}\equiv\left[\begin{array}{cccc|cccc} A_1 & & & & B_1 & & & \\ & A_3 & & & B_2^T & B_3 & & \\ & & \ddots & & & B_4^T & \ddots & \\ & & & A_{n'} & & & \ddots & \\ \hline B_1^T & B_2 & & & A_2 & & & \\ & B_3^T & B_4 & & & A_4 & & \\ & & \ddots & \ddots & & & \ddots & \\ & & & & & & & A_{n''} \end{array}\right]\left[\begin{array}{c} x_1 \\ x_3 \\ \vdots \\ x_{n'} \\ \hline x_2 \\ x_4 \\ \vdots \\ x_{n''} \end{array}\right]=\left[\begin{array}{c} b_1 \\ b_3 \\ \vdots \\ b_{n'} \\ \hline b_2 \\ b_4 \\ \vdots \\ b_{n''} \end{array}\right]\equiv b_{RB}$$

* Received by the editors February 17, 1987; accepted for publication (in revised form) March 31, 1989. This work was supported by Department of Energy contract DE-AC11-88PN38014.

† RT-Mathematics 34F, Bettis Atomic Power Laboratory, P.O. Box 79, West Mifflin, Pennsylvania 15122-0079.

or the condensed form

$$(4) \qquad A_{RB}x_{RB} \equiv \begin{bmatrix} D_R & H \\ H^T & D_B \end{bmatrix} \begin{bmatrix} x_R \\ x_B \end{bmatrix} = \begin{bmatrix} b_R \\ b_B \end{bmatrix}.$$

Eliminating the red unknowns x_R in (4) yields the "reduced system"

$$(5) \qquad A_B x_B \equiv (D_B - H^T D_R^{-1} H)x_B = b_B - H^T D_R^{-1} b_R \equiv c_B.$$

Once the black unknowns x_B are solved for, the red unknowns x_R can be obtained from (4).

The following theorem [5], [9], [20] states the relationship between the coefficient matrices A and A_B of (1)-(5).

THEOREM A. *Let A and A_B be matrices related as in* (1)-(5). *Then the reduced system coefficient matrix A_B satisfies the following*: (a) *A_B is nonsingular whenever A is nonsingular*; (b) *A_B is symmetric positive definite whenever A is symmetric positive definite*; (c) *A_B is strictly diagonally dominant whenever A is strictly diagonally dominant*; *and* (d) *A_B is a Stieltjes matrix whenever A is a Stieltjes matrix.*

See Varga [20] for definitions of strictly diagonally dominant, and Stieltjes matrices.

This paper is concerned with the application of preconditioned polynomial iterative methods [3]-[8], [14]-[16], [18] to the linear systems in (1), (3)-(5). The main idea of preconditioned methods (see § 2 below) is to obtain a good approximation to the coefficient matrix and to use this approximation as a splitting matrix [19] or preconditioner to construct rapidly converging polynomial iterative methods. Of interest here is preconditioning by incomplete block odd-even cyclic reduction [3], [14], [18]. An examination of this successful methods reveals that a significant reduction in computations can be achieved by applying this method to the "reduced system" of (5). Application of preconditioned polynomial acceleration methods to the reduced system is the main focus of this paper.

A special case reduced system that received considerable attention twenty-five to thirty years ago [9], [11], [20] is that corresponding to point red/black partitioning of the system of equations that arises from the standard five-point difference approximation of the two-dimensional elliptic diffusion equation $-\nabla \cdot D\nabla u + \sigma u = S$ over rectangular mesh subdivisions. In this case, D_R and D_R^{-1} in (4) and (5) are diagonal matrices and the reduced system coefficient matrix A_B in (5) is sparse and can be constructed explicitly. Application of preconditioned polynomial iterative methods to such reduced systems is also straightforward.

We are interested in this paper in the more general case when D_R is not a diagonal matrix and when, as a consequence, it is not practical to construct A_B explicitly. It is easily seen that the reduced system coefficient matrix A_B in (5) for the black unknowns has a block tridiagonal structure similar to (1). However, the individual block matrices of the reduced system coefficient matrix A_B are, in general, dense matrices and their explicit construction for large systems would require considerable computer storage. Fortunately, the polynomial iterative methods of interest here require matrix vector multiplications $A_B v = (D_B - H^T D_R^{-1} H)v$, which can be implemented using the individual matrices D_B and H, and the Cholesky factorization of D_R, without the need to explicitly construct A_B and D_R^{-1}.

Several other preconditioned polynomial iterative solution methods have been successfully applied to the block tridiagonal system (1). Because of the complexity involved in constructing or approximating $A_B = D_B - H^T D_R^{-1} H$, these previous applications focused on the choice of preconditioner $Q_B = D_B$ and on forming A_B implicitly. This is the case for the classical reduced system Chebyshev and conjugate gradient methods [12]. Hageman, Luk, and Young [10] have shown that the Chebyshev and conjugate gradient methods applied to the reduced system of (5) converge twice as fast as these methods applied to the systems of (1) and (4). This has led Abu-Shumays [2] to use variants of the incomplete Cholesky factorization to construct new preconditioners Q_B to further improve the convergence of the Chebyshev and conjugate gradient methods applied to this reduced system. The variant of incomplete Cholesky of interest here is incomplete block odd-even cyclic reduction. Below it is shown that this method applied to the block tridiagonal system of (1) leads to a preconditioner Q_B for the reduced system.

One objective of this paper is to assess the effectiveness of preconditioners based on incomplete odd-even cyclic reduction for solving the reduced system. A second objective is to perform numerical experiments and comparisons with other methods on the CYBER 205 vector computer. Elman [7] shares the first objective. He does a good job presenting the details and important computational issues concerning incomplete block cyclic reduction [3], [14], [18]. Elman treats the general case of nonlinear systems and uses a conjugate-gradient like method to solve the preconditioned equations. He performs a comparison on a VAX-8600 in double precision Fortran (55-bit mantissa) and is also concerned with efficient implementation on parallel computers. Axelsson [3], on the other hand, shares the second objective of designing and comparing methods of the one-step Chebyshev type with preconditioners based on incomplete odd-even cyclic reduction as applied to block tridiagonal systems. In contrast to the work of Elman and to the present work, Axelsson, Brinkkemper, and Il'in [3], [4] do not deal with the reduced system.

Section 2 reviews preconditioned Chebyshev and conjugate gradient iterative methods. The algorithms treated here are of the two-step (three-term recursion [12]) Chebyshev type in contrast to the one-step Chebyshev type used by Axelsson and the conjugate gradient formulation heavily used in practice [5], [7], [14], [18]. Section 3 relates preconditioners for the reduced system, the block tridiagonal system and the red/black system. Section 3 then derives equivalence results that generalize the work of Hageman, Luk, and Young [10]. Section 4 reviews the incomplete odd-even cyclic reduction procedure. Section 5 presents two main algorithms for constructing preconditioners based on incomplete block factorization. This section is intended to supplement the fine treatment of Concus, Golub, and Meurant [5]. Finally, § 6 is devoted to numerical experiments.

2. Review of preconditioned polynomial iterative methods.

Consider the linear system

$$Ax = b \tag{6}$$

where the coefficient matrix A is symmetric positive definite. Here, this system stands for any of (1), (3)-(5) and is not to be confused with or restricted to (1). It is assumed that direct methods are not practical. Let Q be a splitting matrix [19] or preconditioner having the same order as A but which is easier to invert than A. It is preferable to select Q to be a good approximation to A. Equation (6) yields

$$x = Gx + k, \qquad G = I - Q^{-1}A, \qquad k = Q^{-1}b. \tag{7}$$

A "basic iterative method" can now be defined from (7) as follows:

$$x^l = Gx^{l-1} + k. \tag{8}$$

This paper focuses on using polynomial methods (called semi-iterative (SI) methods by Varga [20]) to accelerate the basic iterative method of (8). The iteration matrix G

is assumed to be "symmetrizable" in the sense [12] that for some nonsingular matrix W, the matrix $W(I-G)W^{-1}$ is symmetric positive definite. A sufficient but not necessary condition for G to be symmetrizable [12] is that A and Q both be symmetric positive definite. This assumption assures that all eigenvalues μ_i of G are real and less than unity ($\mu_i < 1$). The polynomial methods of interest here include (i) the conjugate gradient (CG) method and (ii) the Chebyshev method, both given by [12]

$$\begin{aligned} x^{n+1} &= \rho_{n+1}[\gamma_{n+1}(Gx^n+k)+(1-\gamma_{n+1})x^n]+(1-\rho_{n+1})x^{n-1} \\ &= \rho_{n+1}[\gamma_{n+1}Q^{-1}\{b-Ax^n\}+x^n]+(1-\rho_{n+1})x^{n-1} \\ &= \rho_{n+1}[\gamma_{n+1}\delta^n+x^n]+(1-\rho_{n+1})x^{n-1}, \end{aligned} \tag{9}$$

where δ^n and r^n are pseudoresidual and residual vectors related by

$$\delta^n \equiv Gx^n+k-x^n = Q^{-1}(b-Ax^n) \equiv Q^{-1}r^n, \qquad r^n \equiv b-Ax^n, \tag{10a}$$

$$r^{n+1} = \rho_{n+1}[r^n-\gamma_{n+1}A\delta^n]+(1-\rho_{n+1})r^{n-1}. \tag{10b}$$

The ρ_n and γ_n in (9) are acceleration parameters. For the conjugate gradient method, the preconditioning matrix Q must be symmetric positive definite, and the acceleration parameters are given by

$$\rho_1 = 1, \qquad \rho_{n+1} = 1\bigg/\left[1-\frac{\gamma_{n+1}}{\gamma_n\rho_n}\frac{(\delta^n, r^n)}{(\delta^{n-1}, r^{n-1})}\right], \qquad \gamma_{n+1} = \frac{(\delta^n, r^n)}{(\delta^n, A\delta^n)}. \tag{11}$$

For the Chebyshev polynomial method, the acceleration parameters satisfy

$$\rho_1 = 1, \qquad \rho_2 = \frac{1}{1-\sigma^2/2}, \qquad \rho_{n+1} = \frac{1}{1-\sigma^2\rho_n/4}, \qquad n \geqq 2, \tag{12a}$$

$$\gamma = \gamma_{n+1} = \frac{2}{2-M(G)-m(G)}, \qquad \sigma = \frac{M(G)-m(G)}{2-M(G)-m(G)} \tag{12b}$$

where $M(G)$ and $m(G)$ are, respectively, the algebraically largest and smallest eigenvalues of G. Note that γ and σ are independent of n.

3. Choice of related preconditioners and equivalence results. Let Q_B be any preconditioner for the reduced system (5). Correspondingly, select a preconditioner Q_{RB} for the red/black system (4) as follows:

$$Q_{RB} \equiv \begin{bmatrix} I & \\ H^TD_R^{-1} & I \end{bmatrix}\begin{bmatrix} D_R & \\ & Q_B \end{bmatrix}\begin{bmatrix} I & D_R^{-1}H \\ & I \end{bmatrix}; \tag{13}$$

and select a preconditioner Q for the original system (1) by

$$Q = U^TQ_{RB}U \qquad (Q_{RB} = UQU^T). \tag{14}$$

Remark 1. Except for the work of Elman in [7], previous Chebyshev and conjugate gradient methods for solving block tridiagonal systems with preconditioners based on incomplete block odd-even cyclic reduction [3], [14], [18] involve preconditioners Q for (1) that satisfy (13) and (14). These methods yield preconditioners Q_B for the reduced system.

Remark 2. Implementation of Chebyshev and conjugate gradient methods for (1), (3)-(5) with preconditioning by block odd-even cyclic reduction is based, in this work, on (9)-(12). This implementation involves matrix vector multiplications $A\delta^n$

and solution of systems $Q\delta^n = r^n$, where A stands for the coefficient matrix in (1)-(5), and Q stands for the corresponding preconditioner. There is no need to construct Q^{-1}, $Q^{-1}A$, or A_B explicitly.

Remark 3. It will become evident below that when applying the Chebyshev and conjugate gradient methods to the large systems in (1) and (4) with preconditoners Q and Q_{RB} of (13) and (14), the iterates $\{x_B^l\}$ for the black unknowns remain independent of the iterates $\{x_R^l\}$ for the red unknowns at all stages of the iteration. In essence, the computation of the x_R^l iterates is wasteful. Furthermore, with a proper choice of initial guess, approximately half of the residuals involved in such computations are zeros and need not be computed. This suggests that a significant reduction in computations can be realized by applying the Chebyshev and conjugate gradient methods with preconditioner Q_B to the reduced system (5).

The objective now is to establish some equivalence results.

THEOREM 1. *The preconditioners Q and Q_{RB} of* (13) *and* (14) *are symmetric positive definite if and only if the reduced system preconditioner Q_B of* (13) *is symmetric positive definite.*

Proof. The proof concerning Q_{RB} and Q_B follows directly from (13) and the fact that D_R in this equation is symmetric positive definite. The desired result for Q and Q_B then follows from (14). □

THEOREM 2. *Consider any polynomial iterative method applied to the block tridiagonal system in* (1) *and to the red/black system of* (2)-(4). *Let A in* (1) *be symmetric positive definite. Then the choice of preconditioners Q and Q_{RB} given by* (13) *and* (14) *leads to a decoupling of iterates of the black unknowns $\{x_B^l\}$ from iterates of the red unknowns $\{x_R^l\}$ in the sense that $\{x_B^l\}$ can be constructed independently of $\{x_R^l\}$ throughout the iteration.*

Proof. Consider first the preconditioner Q_{RB} of (13). Its inverse is given by

$$Q_{RB}^{-1} = \begin{bmatrix} I & -D_R^{-1}H \\ 0 & I \end{bmatrix} \begin{bmatrix} D_R^{-1} & \\ & Q_B^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -H^T D_R^{-1} & I \end{bmatrix}. \tag{15}$$

Consequently, the iteration matrix for (4) is

$$G_{RB} = I - Q_{RB}^{-1}A_{RB} = \begin{bmatrix} 0 & -D_R^{-1}H(I - Q_B^{-1}A_B) \\ 0 & (I - Q_B^{-1}A_B) \end{bmatrix} = \begin{bmatrix} 0 & -D_R^{-1}HG_B \\ 0 & G_B \end{bmatrix}, \tag{16}$$

and the corresponding basic iterative method $x_{RB}^{n+1} = G_{RB}x_{RB}^n + k_{RB}$ is

$$\begin{bmatrix} x_R^{n+1} \\ x_B^{n+1} \end{bmatrix} = \begin{bmatrix} 0 & -D_R^{-1}HG_B \\ 0 & G_B \end{bmatrix} \begin{bmatrix} x_R^n \\ x_B^n \end{bmatrix} + \begin{bmatrix} k_R \\ k_B \end{bmatrix}, \tag{17}$$

which implies that

$$x_B^{n+1} = G_B x_B^n + k_B. \tag{18}$$

Thus, for the basic iterative method of (18), the black iterates x_B^{n+1} remain independent of the red iterates x_R^n. Since any polynomial iterative method [12], [20] generates iterates that are linear combinations of those in (17) and (18), it follows from (18) that linear combinations of $\{x_B^n\}$ remain independent of linear combinations of $\{x_R^l\}$. This proves that Theorem 2 is valid for any polynomial method applied to the red/black systems in (4), with preconditioner Q_{RB} given by (13). The fact that A_{RB} and Q_{RB} are related to A and Q by a permutation or reordering of the unknowns implies that the theorem also holds for any polynomial method applied to (1) with preconditioner Q

of (13) and (14). Note from (1)-(4) that x_B is the vector of even-numbered (black) unknowns in x in (1). $\square$

Theorem 2 and the expression for the iteration matrix in (17) and (18) suggest that computational economy can be realized by applying polynomial iterative methods to the reduced system with preconditioner Q_B in preference to applying such methods, as in [3], [14], [18], with preconditioner Q of (13) and (14) to the larger system of (1). Once x_B is solved for, the solution x_R can be obtained from (4). Additional equivalence results are presented in the balance of this section for the sake of completeness.

LEMMA 1. *Let Q, Q_{RB}, and Q_B be as given in* (13) *and* (14). *Then the spectrum of eigenvalues of the red/black system iteration matrix $G_{RB} = I - Q_{RB}^{-1}A_{RB}$ is the same as the spectrum of eigenvalues of the full system iteration matrix $G = I - Q^{-1}A$. This spectrum includes, as a subset, the spectrum of eigenvalues of the reduced system iteration matrix $G_B = I - Q_B^{-1}A_B$.*

Proof. Since $A_{RB} = UAU^T$, $Q_{RB} = UQU^T$, and $UU^T = I$, I the identity matrix, it is straightforward to show that

$$G_{RB} = UGU^T, \qquad G = U^T G_{RB} U. \tag{19}$$

Since U is an orthogonal matrix, it follows that the matrices G_{RB} and G have identical eigenvalues. The expression for G_{RB} in (16) implies that all eigenvalues of the reduced system iteration matrix G_B are necessarily eigenvalues of the red/black system iteration matrix G_{RB} of this equation. This completes the proof. $\square$

Equation (16) implies that G_{RB} has a set of zero eigenvalues that corresponds to the zero diagonal block. G_B need not have zero eigenvalues. To ensure that the eigenvalues of G_B and G_{RB} are real, it is assumed that the splitting matrices Q_B and Q_{RB} are symmetric positive definite and thus G_B and G_{RB} are symmetrizable. This assumption is possible because of Theorem 1.

The equivalence results given in Theorems 3-5 below (see also [2]) generalize the work of Hageman, Luk, and Young [10].

THEOREM 3. *Let Q_{RB} and Q_B of* (13) *be symmetric positive definite preconditioners. Also let $(x_{RB}^n)^T = ([x_R^n]^T, [x_B^n]^T)$ denote the conjugate gradient iterates for red/black system* (4) *with preconditioner Q_{RB}; and let $\{\tilde{x}_B^n\}$ denote the conjugate gradient iterates for the reduced system* (5) *with preconditioner Q_B. If the initial iterates satisfy $x_B^0 = \tilde{x}_B^0$, $x_R^0 = D_R^{-1}b_R - D_R^{-1}H\tilde{x}_B^0$ where $\tilde{x}_B^0$ is arbitrary, then $x_B^n = \tilde{x}_B^n$ for all n. However, if $x_B^0 = \tilde{x}_B^0$ but $x_R^0 \neq D_R^{-1}b_R - D_R^{-1}H\tilde{x}_B^0$, then x_B^n and $\tilde{x}_B^n$ need no longer be identical.*

The proof is omitted for brevity. One step in the proof [2] shows that whenever $x_B^0 = \tilde{x}_B^0$ and $x_R^0 = D_R^{-1}b_R - D_R^{-1}H\tilde{x}_B^0$, the residual vector for the red/black system satisfies

$$r_{RB}^n = \begin{bmatrix} 0 \\ r_B^n \end{bmatrix}, \qquad r_B^n = c_B - A_B x_B^n \quad \text{for all } n. \tag{20}$$

Thus, in this case at each step of the iteration approximately half of the elements of the residual r_{RB}^n must vanish and need not be computed.

A weaker result holds for Chebyshev iterations.

THEOREM 4. *Let Q, Q_{RB}, and Q_B of* (13) *and* (14) *be symmetric positive definite preconditioners, respectively, for the block tridiagonal system* (1), *the corresponding red/black system* (4), *and the reduced system* (5). *Assume further that the coefficient matrix A of* (1) *is symmetric positive definite. Then the rate of convergence of the optimum Chebyshev polynomial method applied to the reduced system is at least as fast as the rate* *of convergence of the optimum Chebyshev method applied to either the block tridiagonal system or the red/black system.*

Proof. Theorem A and (2) imply that A_{RB} and A_B are necessarily symmetric positive definite. Since Q, Q_{RB}, Q_B, A, A_{RB}, and A_B are all symmetric positive definite, it follows that all eigenvalues of the iteration matrices $G = I - Q^{-1}A$, $G_{RB} = I - Q_{RB}^{-1}A_{RB}$, and $G_B = I - Q_B^{-1}A_B$ are real and less than unity [12]. Lemma 1 and its proof established that (a) G and G_{RB} have identical eigenvalues, (b) the nonzero eigenvalues of the iteration matrices G, G_{RB}, and G_B are identical, and (c) that G and G_{RB} must have a zero eigenvalue of multiplicity at least equal to the order of the red unknowns. Since zero may or may not be in the range of eigenvalues of G_B, it follows that the ranges of eigenvalues of G_B, G_{RB}, and G satisfy

$$[m(G_B), M(G_B)] \subset [m(G_{RB}), M(G_{RB})] \equiv [m(G), M(G)]. \tag{21}$$

It then follows from (21), (9), (12a), (12b), and from Hageman and Young [12] that the rates of convergence of the optimal Chebyshev method applied to the block tridiagonal system and applied to the red/black system are identical. Furthermore, these rates of convergence are at best as fast as the rates of convergence of the optimal Chebyshev method applied to the reduced system. □

For most practical problems, the ranges of eigenvalues of G_B, G_{RB}, and G in (21) are expected to be the same. For such problems, the Chebyshev iterates x_B^n for the black unknowns of the block tridiagonal system are identical to the Chebyshev iterates $\tilde{x}_B^n$ for the reduced system provided that $x_B^0 = \tilde{x}_B^0$.

4. Exact and approximate block odd–even cyclic reduction. Odd-even cyclic reduction is an effective method for factoring and solving relatively small block tridiagonal systems on a vector computer. The method is not practical for very large block tridiagonal systems because of its excessive storage requirements. On the other hand, approximate or so-called "incomplete" odd-even cyclic reduction has been shown to be effective [3], [14], [18] for constructing preconditioners for accelerating the convergence of the polynomial iterative solution of such systems. The basic steps of complete and incomplete odd-even cyclic reduction are summarized in this section. Additional details are supplied in [1], [3], [7], [13], [14], [18], and in § 5 below.

Complete odd-even cyclic reduction of a block tridiagonal symmetric positive definite matrix A, as in (1), is a global factorization represented symbolically by [1, p. 36]

$$\begin{aligned} A &= P_1 A_{RB} P_1^T = P_1 L_1 D_1 L_1^T P_1^T = P_1 L_1 P_2 L_2 D_2 L_2^T P_2^T L_1^T P_1^T \\ &= P_1 L_1 P_2 L_2 \cdots P_r L_r D_r L_r^T P_r^T \cdots L_2^T P_2^T L_1^T P_1^T, \qquad P_1 = U^T, \end{aligned} \tag{22}$$

and obtained by a sequence of permutation P_i and factorization $L_i D_i L_i^T$ steps. The first step consists of (a) applying the transformation $A_{RB} = UAU^T$ to A to obtain the red/black form A_{RB} shown in (4), and (b) factoring A_{RB} as follows:

$$A_{RB} = \begin{bmatrix} D_R & H \\ H^T & D_B \end{bmatrix} = \begin{bmatrix} I & \\ H^T D_R^{-1} & I \end{bmatrix} \begin{bmatrix} D_R & \\ & A_B \end{bmatrix} \begin{bmatrix} I & D_R^{-1} H \\ & I \end{bmatrix} \equiv L_1 D_1 L_1^T, \tag{23}$$

where A_B, the lower diagonal block part of D_1, is the coefficient matrix of the reduced system.

It can be shown that A_B is approximately half the size of and has the same block tridiagonal form as the matrix A in (1). Thus the process of permutation in (2) and the first part of (22) to transform the matrix A to the red/black A_{RB} form of (4), followed by factorization as in (23), can be applied to the lower diagonal block A_B of D_1 as indicated symbolically in the top part of (22). The matrices L_2 and D_2 in (22) that result from this process have the form

$$L_2 = \begin{bmatrix} I & \\ & \begin{bmatrix} I & \\ H_2^T D_{R2}^{-1} & I \end{bmatrix} \end{bmatrix}, \qquad D_2 = \begin{bmatrix} D_r & \\ & \begin{bmatrix} D_{r2} & \\ & A_B^{(2)} \end{bmatrix} \end{bmatrix}, \tag{24}$$

where D_R, D_{R2} are block diagonal matrices. Here D_{R2} is approximately half the size of D_R, $A_B^{(2)}$ is a block tridiagonal second stage reduced system matrix approximately one fourth the size of the matrix A, and H_2^T is a matrix having the same block structure of the matrix H^T in (3) and (4), except that H_2^T is approximately half the size of H^T. The permutation and factorization process can now be applied to the lower block diagonal part $A_B^{(2)}$ of D_2 and to further smaller and smaller block tridiagonal submatrices $A_B^{(i)}$ until a final single block matrix $A_B^{(r)}$ is obtained. In other words, the process above can be continued for r steps whereby D_r in (22) is a block diagonal matrix whose blocks have the same size as the diagonal blocks of A. This is a brief description of complete block odd-even cycle reduction. Once the factorization in (22) is completed, the solution of (1) for different right-hand sides can be obtained by a series of permutation and forward elimination steps to obtain smaller and smaller reduced systems, followed by a solution of a single block equation for the smallest reduced system, and finally followed by successive permutation and backward elimination steps to recover the remaining unknowns.

As mentioned above, block odd-even cyclic reduction applied to (1) is not practical for very large systems because of excessive storage requirements. For example, although the initial block matrices A_i and B_i in (1) are often sparse, subsequent block matrices of the reduced systems are, in general, dense matrices that must be stored in order to complete the above factorization and solution procedure. Consequently, we are led to incomplete odd-even cyclic reduction, in which the factorization is only approximately carried out. Such approximations, which yield preconditioning matrices, can be constructed within moderate storage requirements based on the following: Heller [13] proved that under conditions of diagonal dominance, the off-diagonal blocks of the successively smaller reduced systems decay quadratically to zero. Consequently, a reasonable approximation to the matrix A can be obtained by early termination of block odd-even cyclic reduction procedure by carrying out only a few J ($J = 1, 2, \cdots$) permutations and factorization steps, and by ignoring the last set of off-diagonal blocks. In other words, r in (22) is set equal to J and the off-diagonal blocks of D_J (of $A_B^{(J)}$) are ignored.

A second approximation of block odd-even cyclic reduction can be obtained by imposing a preselected sparsity pattern on the successive individual tridiagonal blocks of D_i (of $A_B^{(i)}$) in (22) by neither computing nor storing matrix elements outside the selected pattern [3], [7], [14], [18]. Rodrigue and Wolitzer [18] have generalized the results of Heller [13] to the special case when the matrix A of (1) is a Stieltjes matrix. They have proved that in this case, the resulting reduced system matrices are also Stieltjes matrices and that the off-diagonal blocks of the successively smaller reduced systems for this incomplete odd-even reduction also decay quadratically to zero. The fact that the reduced matrices $A_B^{(i)}$ are Stieltjes matrices and thus are positive definite implies numerical stability of the incomplete block odd-even cyclic reduction factorization.

Kershaw [14] was the first to combine both aspects of incomplete block odd-even cyclic reduction: (i) imposing a sparsity pattern, and (ii) early termination of the procedure. He applied his algorithms to block tridiagonal systems where the coefficient matrices are positive definite but are not Stieltjes matrices. In this more general situation, a modification of the diagonal elements of the reduced systems may be necessary to ensure positive definiteness and numerical stability [8], [14].

5. Algorithms for incomplete block odd–even cyclic reduction. The incomplete block odd-even cyclic reduction of interest here involves both termination of the cyclic reduction process after J reduction steps and at each step imposing a prescribed sparsity pattern and approximating the resulting reduced system matrix $A_B^{(i)}$. For the rest of this paper, it is assumed for illustration that each block of the initial block tridiagonal matrix A of (1) is a tridiagonal matrix. It is also assumed that each block of a successive block tridiagonal reduced system matrix $A_B^{(i)}$ is approximated by a tridiagonal matrix. Elman [7] approximates the block tridiagonal matrix $A_B^{(J)}$ for the last reduction step by an incomplete LU-type factorization. The treatment in [7] is advantageous but requires additional computer storage. The off-diagonal blocks of the last reduced system matrix $A_B^{(J)}$ are ignored here as in [14], [18]. The incomplete block odd-even cyclic reduction described there leads to the following preconditioner approximating (22):

$$(25) \qquad Q = P_1 Q_{RB} P_1^T = P_1 L_1 P_2 \tilde{L}_2 \cdots P_J \tilde{L}_J D_J \tilde{L}_J^T P_J^T \cdots \tilde{L}_2^T P_2^T L_1^T P_1^T.$$

L_1 and L_1^T in (22), (23), and (25) are not approximated here. Moreover, these matrices are not constructed explicitly. Instead H and a factored form of D_R (either LDL^T or odd-even cyclic reduction of D_R as in [1]) are retained (cf. (23)). Similarly, the $\tilde{L}_i$ are not constructed explicitly but their components $\tilde{H}_i$ and factored forms of $\tilde{D}_i$'s (see (24)) are retained. Note that each block of A, and of subsequent approximations to the reduced system matrices $A_B^{(i)}$, is a tridiagonal matrix. It follows from (1)–(4) and similar transformations that the matrices H and $\tilde{H}_i$ are block bidiagonal matrices where each block is itself a tridiagonal matrix. Furthermore, the size of $\tilde{H}_i$ is approximately $1/2^i$ times the size of H.

Total computer storage required for $H, \tilde{H}_2, \cdots, \tilde{H}_J$ is approximately $6N(1-1/2^J)$ where N is the order of the initial system of (1) (values of N of interest vary between 40,000 and 250,000; $N \simeq 60{,}000$ is used for the numerical experiments reported below). Approximately $3N$ storage locations are required for D_B, and the LDL^T factorization of $D_R, D_{R2}, \cdots, D_{RJ}$. An additional N storage locations would be needed for odd-even cyclic reduction of the tridiagonal matrices $D_R, D_{R2}, \cdots, D_{RJ}$. (See [1] for vectorization and the results of factorizing independent tridiagonal systems on the CYBER 205.) Additional working storage to accomplish the global incomplete block odd-even cyclic reduction is neglected here in view of other storage requirements of the polynomial methods of interest.

Equations (25) and (13) combined with the expression for L_1 in (23) yield the preconditioner Q_B for the reduced system.

The form $A_B = D_B - H^T D_R^{-1} H$ of (5) is typical of the form of the successively smaller reduced systems matrices $A_B^{(i)}$, and approximating each essentially involves a term analogous to

$$(26) \qquad H^T D_R^{-1} H = H^T L^{-T} D^{-1} L^{-1} H,$$

where LDL^T is the Cholesky factorization of the tridiagonal matrix D_R. The approximations of interest affect the entries of, but not the structure or sparsity of, various factors in (25); thus, such approximations do not affect the cost per iteration of any preconditioned polynomial iterative method. Such approximations would, however, affect the number of iterations needed to achieve a desired accuracy. The approximation of the terms in (26) can be accomplished by several algorithms described in [5] by first approximating D_R^{-1} or L^{-1}. In particular, the polynomial approximations in [5] can be considerably improved by exploiting matrix symmetry and properties of eigenvalues of corresponding iteration matrices [2]. The approximations favored by us are given below.

5.1. Approximation from Cholesky factors. Rodrigue and Wolitzer [18] suggest starting from the Cholesky factorization

$$D_R = LDL^T, \qquad D_R^{-1} = L^{-T}D^{-1}L^{-1}, \tag{27}$$

where D is a diagonal matrix and where L is a unit lower bidiagonal matrix. Note that L can be written as

$$L = I + l, \tag{28}$$

where I is the identity matrix and l is a matrix having zero elements everywhere except for its lower diagonal elements $l_{i+1,i}$ that coincide with the corresponding elements of L. It follows [18, Appendix II] that

$$L^{-1} = \sum_{i=0}^{n-1} (-l)^i, \tag{29}$$

where n is the order of L. The desired approximation to L^{-1} is readily obtained by truncating the series in (29). Keeping two or three terms in the right-hand side of (29) is sufficient for the present work [5], [18]. The procedure adopted in this work to implement the above approximation is as follows. The Cholesky factorization of the tridiagonal matrix D_R (of order $\sim N/2$) is obtained by the parallel line vector method in [1] and requires N words of storage plus $N/2$ words of work storage. For the numerical results reported below, the series in (29) is truncated to three terms. The matrix product $D^{-1/2}L^{-1}H$ is accomplished in two steps by first multiplying $D^{-1/2}L^{-1}$ by the block diagonal part of H and then by the lower block subdiagonal part of H. Note that these separate parts of H can be viewed as tridiagonal matrices. The multiplication of each part of H by $D^{-1/2}L^{-1}$ requires approximately $5(N/2)$ storage locations (for a total of $5N$ additional storage locations), is vectorizable by multiplication by diagonals, and requires nine vector multiplies and two vector adds, each of vector length approximately $N/2$. A vector square root of length $N/2$ is also applied to D^{-1}. Construction of the sparse block tridiagonal approximation to A_B is then accomplished by the matrix multiplication $(D^{-1/2}L^{-1}H)^TD^{-1/2}L^{-1}H$ with L^{-1} replaced by the first three terms in (29). This matrix multiplication involves an additional 16 vector multiplies and 16 vector adds for the block diagonal part, and 12 vector multiplies and nine vector adds for the block subdiagonal part; each vector being of length $N/2$. In summary, given the LDL$^\text{T}$ Cholesky factorization of D_R, the above sparse approximation to the first reduced system matrix A_B requires 46 vector multiplies, 29 vector adds, and one vector square root each of length $N/2$. The vector length decreases by a factor of two for each subsequent stage of the incomplete reduction. During the first few stages of the reduction, $N/2 \simeq 30{,}000$, and so the operations are carried out at near optimal vector speed.

Other implementations [3], [7] first construct a banded approximation to D_R^{-1} and then evaluate $H^TD_R^{-1}H$. Elman [7] constructs an approximation to D_R^{-1} that involves seven diagonals.

5.2. New approximations. The objective here is to approximaate $D_R^{-1}H$ of (5) and (26) directly without first approximating D_R^{-1} or its Cholesky factor L^{-1}. This is in contrast to the treatment above and in [3], [5], [7], [14], and [18]. The concepts involved will be illustrated by a simple example based on approximating $D_R^{-1}T$ where T is tridiagonal. It is convenient to introduce the following definition.

DEFINITION 1. Let $A=(a_{ij})$ be a square matrix. $\mathrm{Sp}_l(A)$ is defined as the sparse matrix whose elements are the same as those of A for $|i-j|\leqq l$ and are zero otherwise.

Thus, irrespective of the structure of A, $\mathrm{Sp}_l(A)$ is a banded matrix. In particular $\mathrm{Sp}_1(A)$ is the tridiagonal matrix that is equal to the tridiagonal part of A. Clearly, $A\equiv \mathrm{Sp}_{n-1}(A)$ for any n by n matrix A.

Consider now this simplified problem. Find the banded matrix solution Φ to the matrix equation

$$\mathrm{Sp}_1(D_R\Phi)=T \tag{30}$$

where D_R and T are tridiagonal matrices and D_R is symmetric positive definite. The matrix Φ is regarded as an approximation to $D_R^{-1}T$.

Equation (30) is not a standard equation. For illustration it suffices to consider the special case of Φ being a tridiagonal matrix. In this case, (30) may be written explicitly as follows:

$$\begin{aligned}
&\mathrm{Sp}_1\left(\begin{bmatrix} d_1 & c_1 & & & \\ c_1 & d_2 & c_2 & & \\ & c_2 & \ddots & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ & & & c_{n-1} & d_n \end{bmatrix}\begin{bmatrix} \beta_1 & \alpha_1 & & & \\ \gamma_1 & \beta_2 & \alpha_2 & & \\ & \gamma_2 & \ddots & \ddots & \\ & & \ddots & \ddots & \alpha_{n-1} \\ & & & \gamma_{n-1} & \beta_n \end{bmatrix}\right) \\
&\qquad = \begin{bmatrix} s_1 & r_1 & & & \\ t_1 & s_2 & r_2 & & \\ & t_2 & \ddots & \ddots & \\ & & \ddots & \ddots & r_{n-1} \\ & & & t_{n-1} & s_n \end{bmatrix},
\end{aligned} \tag{31}$$

which is equivalent to the following coupled system:

$$d_{i-1}\alpha_{i-1}+c_{i-1}\beta_i=r_{i-1}, \qquad i=2,\cdots,n, \tag{32a}$$

$$c_{i-1}\alpha_{i-1}+d_i\beta_i+c_i\gamma_i=s_i, \qquad i=1,\cdots,n, \tag{32b}$$

$$c_i\beta_i+d_{i+1}\gamma_i=t_i, \qquad i=1,\cdots,n-1. \tag{32c}$$

Eliminating the α and γ unknowns in (32a)-(32c) yields

$$\begin{aligned}
&\left\{d_i-\frac{c_{i-1}^2}{d_{i-1}}-\frac{c_i^2}{d_{i+1}}\right\}\beta_i=s_i-\frac{r_{i-1}c_{i-1}}{d_{i-1}}-\frac{t_ic_i}{d_{i+1}}, \qquad i=1,\cdots,n, \\
&c_0=c_n=0 \quad \text{and} \quad d_0=d_{n+1}=1.
\end{aligned} \tag{33}$$

The assumption that D_R is strictly diagonally dominant ensures the numerical stability of evaluating β_i from (33), and of evaluating α_i and γ_i from β_i and (32a), (32c). Note that these computations are vectorizable with vector length n or $n-1$. Specifically the right-hand sides of (33) require one vector divide to evaluate $1/d_i$, two vector multiplies to evaluate c_i/d_i and c_i/d_{i+1} ($1/d_i$, c_i/d_i, and c_i/d_{i+1} are temporarily saved for the balance of the computations), two vector multiplies (by r_i and t_i), and two vector adds (subtracts). The coefficients of β_i on the left-hand side of (33) can be evaluated with two additional vector multiplies and two additional vector adds. The β_i can then be computed by one final vector divide. Subsequent evaluation of α_i and γ_i requires two vector multiplies and one vector add. In summary, the computation of Φ requires (in the present case) a total of 10 vector multiplies, six vector adds, and two vector divides.

While the discussion above centered on a tridiagonal part of H, it is straightforward to deduce from (3) that $D_R^{-1}H$ can be approximated by first letting T stand for the odd tridiagonal blocks B_1, B_3, etc., comprising the diagonal blocks of H, and then letting T stand for the even tridiagonal blocks B_2^T, B_4^T, etc., comprising the lower subdiagonal blocks of H. Once $D_R^{-1}H$ is approximated, (5) and (26) yield an approximation to A_B. This computation for the more general case of Φ as a pentadiagonal matrix can be shown to require 45 vector multiplies, 21 vector adds, and four vector divides, each of length $n/2$. Thus this computation compares favorably with the approximation of § 5.1 based on the Cholesky factors. The latter approximation requires 46 vector multiplies, 29 vector adds, and one vector square root in addition to requiring the evaluation of the Cholesky factorization $D_R = LDL^T$. The new approximations require less storage to implement and are preferred to those in § 5.1 for larger problems where conserving computer storage is vital.

6. Numerical comparison. Several preconditioned Chebyshev and conjugate gradient methods were implemented on a CYBER 205 using the experimental program DXY [1], [2], which solves the following Diffusion equation in XY rectangular geometry

$$-\nabla \cdot D\nabla u + \sigma u = S, \tag{34}$$

with Dirichlet or Neumann boundary conditions. Here u is particle density, D is a diffusion coefficient, σ is a cross section, and S is a source term. A finite-difference approximation of this diffusion equation over triangulated parallelogram mesh subdivisions leads to a block tridiagonal linear system of the form of (1), where the diagonal blocks A_i are tridiagonal matrices and the off-diagonal blocks B_i are lower bidiagonal matrices. Here each diagonal block describes the couplings between the unknowns on a mesh line along the x-direction.

Vectorization of the various solution method options in the DXY program is achieved (a) by applying CYBER 205 vector syntax where possible; (b) by implementing the hyperline concept [1], [2] of combining all the red (odd-numbered) lines into a single red hyperline and all the black (even-numbered) lines into a black hyperline; and (c) by repeatedly using a vectorized odd-even cyclic reduction algorithm [1] to solve the tridiagonal systems for the hyperlines associated with the application of the Chebyshev and conjugate gradient methods to (1) and (5). The hyperline approach for solving sets of independent tridiagonal systems is used throughout this study for consistency. An alternative vectorization in which independent tridiagonal systems are solved in parallel [1] may be more efficient for large problems, but is not expected to alter the relative performance of the various methods and so is not considered here.

In the present work, the following relative error measure [12, p. 71] is used as a stopping criterion:

$$\frac{1}{1-M_E} \max_i |\delta_i^n / x_i^n| < \zeta \tag{35}$$

where ζ is the prescribed error tolerance and M_E is an estimate of $M(G)$ obtained by one of the adaptive procedures of Hageman and Young [12, Appendices A and B]. Here δ_i is the ith element of the vector δ defined by (10a).

The implementation of the Chebyshev method in DXY relies on subroutines CCSI and CHEBY of Hageman and Young [12] to adaptively generate and update the acceleration parameters in (9). These subroutines also check for convergence and provide a stopping criterion satisfying (35). In contrast, the conjugate gradient method as implemented in DXY requires the user to put M_E for application in the stopping criterion of (35). For the problems considered here, the Chebyshev method was applied first and estimates M_E obtained from this method were then passed to the conjugate gradient method. It is noted that the stopping criterion specified in (35) is conservative for the conjugate gradient method. It is also noted that the conjugate gradient algorithm is capable of computing M_E on its own in view of the connection with the Lanczos method. The actual computation is not expected to affect the relative performance of the various methods.

Construction of preconditioners based on incomplete block odd–even cyclic reduction was described in §§ 4 and 5 above. In particular, two ways to approximate the reduced system coefficient matrix are given in §§ 5.1 and 5.2. These approximations are repeated at each step of the block odd–even cyclic reduction algorithm. The odd–even cyclic reduction algorithm is terminated after J reduction steps to yield an incomplete factorization approximating the first reduced system coefficient matrix A_B. The approximations in §§ 5.1 and 5.2 were numerically tested. These approximations led to comparable run times on the CYBER 205 for the preconditioned conjugate gradient and Chebyshev methods whenever the number of terms retained for the inverse of the Cholesky factor L in (29) was the same as the bandwidth of the matrix Φ defined above. Results using the approximation of § 5.2 are omitted for brevity. The approximations based on § 5.1 are used in the numerical examples reported below for possible comparison with work of other authors [14], [18].

The methods compared in this paper are designated as follows:

CG	The standard conjugate gradient method [12] applied to (1).
PT-SI	Chebyshev method of (9) applied to (1) with a preconditioner $Q = D$, the diagonal matrix whose elements are the diagonal elements of the coefficient matrix A of (1).
PT-CG	Same as PT-SI but with the conjugate gradient method instead of the Chebyshev polynomial method.
P-SI-J	Chebyshev method of (9) applied to (1) with a preconditioner based on a J-step, $J = 1, 2, \cdots$, incomplete block odd–even cyclic reduction algorithm as in (25). The diagonal and off-diagonal blocks of the resulting reduced system coefficient matrix at each step of the factorization are chosen to be tridiagonal matrices.
P-CG-J	Same as P-SI-J but with the conjugate gradient method instead of the Chebyshev method.
CCSI-H	Red/black line cyclic Chebyshev (semi-iterative) method [1], [12] applied to the red/black system (4) with the hyperline method [1] used for inverting the diagonal blocks D_R and D_B.
RS-SI	Chebyshev method of (9) applied to the reduced system (5) with a preconditioner $Q_B = D_B$ corresponding to the black unknowns in (4).
RS-CG	Same as RS-SI but with the conjugate gradient method instead of the Chebyshev method.
P-RS-SI-J	Chebyshev method of (9) applied to the reduced system (5) with

a preconditioner Q_B based on a J-step incomplete block odd-even cyclic reduction algorithm applied to the original block tridiagonal system (1). The diagonal and off-diagonal blocks of the reduced system coefficient matrices that result at each step of the factorization are selected here to be tridiagonal matrices.

P-RS-CG-J Same as P-RS-SI-J but with the conjugate gradient method instead of the Chebyshev method.

The various methods were tested for several problems including those illustrated in Tables 1-4 and in Fig. 1.

Numerical results summarized in part in the various tables indicate the following:

(i) The time ratios for P-SI-J over P-RS-SI-J and P-CG-J over P-RS-CG-J are shown in Tables 1-4 to lie between 1.51 and 2.10. These results suggest that preconditioned reduced system polymonial methods with preconditioners based on incomplete block odd-even cyclic reduction are computationally superior to the corresponding

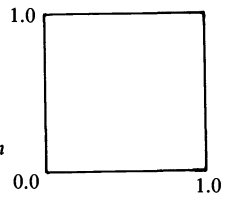


TABLE 1
CYBER 205 *run time (sec) for various methods applied to the model problem with* 90° *angle.*

$D = .01$, $\sigma = 1.0$, $S = 1.0$

Uniform mesh	160×400			256×250			400×160		
Method	Iter	M_E	Time	Iter	M_E	Time	Iter	M_E	Time
PT-SI	379	.9997	7.63	337	.9995	6.76	379	.9997	7.58
P-SI-1	177	.9985	8.42	119	.9966	5.70	85	.9935	4.08
P-SI-2	94	.9946	5.33	75	.9918	4.29	69	.9902	3.94
P-SI-3	61	.9855	3.78	66	.9893	4.09	69	.9900	4.25
CG	379	.99997	9.68	317	.99997	8.08	379	.99997	9.63
PT-CG	331	.9997	8.26	263	.9995	6.54	331	.9997	8.22
P-CG-1	158	.9985	8.45	94	.9966	5.05	61	.9935	3.28
P-CG-2	71	.9946	4.68	48	.9918	3.04	42	.9902	2.65
P-CG-3	35	.9862	2.40	42	.9893	2.87	42	.9900	2.85
CCSI-H	183	.9996	4.02	118	.9990	2.60	82	.9977	1.81
RS-SI	174	.9993	3.88	109	.9981	2.44	70	.9953	1.58
P-RS-SI-2	98	.9946	3.25	75	.9918	2.50	56	.9902	1.88
P-RS-SI-3	54	.9862	2.07	64	.9893	2.44	55	.9900	2.10
P-RS-SI-4	58	.9805	2.35	62	.9892	2.52	55	.9900	2.23
RS-CG	171	.9993	4.52	101	.9981	2.68	66	.9953	1.76
P-RS-CG-2	74	.9946	2.66	47	.9918	1.73	41	.9902	1.51
P-RS-CG-3	38	.9862	1.59	40	.9893	1.68	43	.9900	1.78
P-RS-CG-4	30	.9805	1.35	40	.9892	1.79	43	.9900	1.88
Ratios									
P-SI-J/P-RS-SI-J		$J=2$	1.64			1.72			2.10
		$J=3$	1.83			1.68			2.02
P-CG-J/P-RS-CG-J		$J=2$	1.76			1.76			1.75
		$J=3$	1.51			1.71			1.60

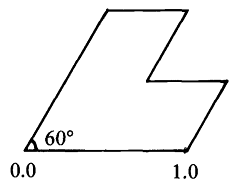


TABLE 2
CYBER 205 *run times (sec) for various methods applied to the* L-*shaped domain with* 60° *angle.*

$D = .01$, $\sigma = 1.0$, $S = 1.0$

Uniform mesh	160×400			256×250			400×160		
Method	Iter	M_E	Time	Iter	M_E	Time	Iter	M_E	Time
PT-CG	459	.9997	11.51	303	.9995	7.61	459	.9997	11.45
P-CG-1	215	.9988	11.35	136	.9972	7.20	87	.9945	4.61
P-CG-2	106	.9956	6.55	80	.9945	4.96	78	.9939	4.82
P-CG-3	56	.9897	3.76	73	.9939	4.87	78	.9938	5.17
CCSI-H	208	.9997	4.54	129	.9992	2.83	85	.9981	1.87
RS-CG	228	.9994	6.03	135	.9985	3.58	94	.9963	2.49
P-RS-CG-2	106	.9956	3.77	80	.9945	2.87	79	.9939	2.82
P-RS-CG-3	56	.9897	2.29	74	.9939	3.00	81	.9939	3.25
P-RS-CG-4	50	.9870	2.18	74	.9938	3.18	81	.9939	3.43
Ratios									
P-CG-J/P-RS-CG-J		$J=2$	1.74			1.73			1.71
		$J=3$	1.64			1.62			1.59

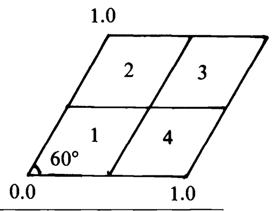


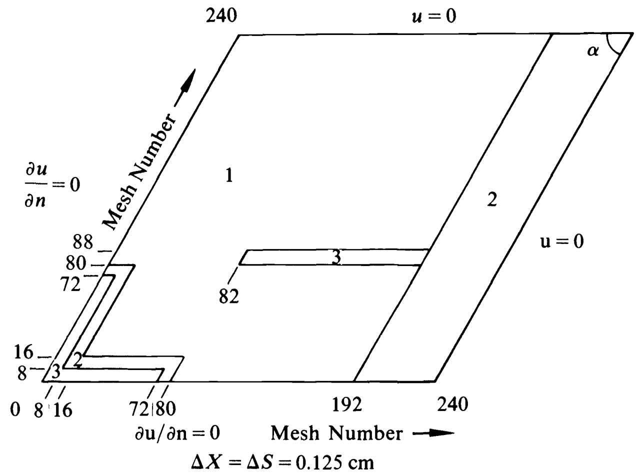
TABLE 3
CYBER 205 *run time (sec) for various methods applied to a four region problem with* 60° *angle.*

$\sigma = S = 1.0$ throughout. $D_1 = .0025$, $D_2 = .025$, $D_3 = .25$, $D_4 = 2.5$

Uniform mesh	160×400			256×250			400×160		
Method	Iter	M_E	Time	Iter	M_E	Time	Iter	M_E	Time
PT-CG	791	.9999	19.78	527	.9999	13.18	802	.9999	19.96
P-CG-1	363	.9997	19.12	224	.9994	11.82	142	.9987	7.49
P-CG-2	177	.9990	10.87	133	.9987	8.19	126	.9984	7.73
P-CG-3	92	.9976	6.10	123	.9986	8.12	126	.9984	8.29
CCSI-H	583	.9999	12.62	373	.9998	8.08	245	.9996	5.31
RS-CG	405	.9999	10.66	238	.9996	6.27	152	.9991	4.00
P-RS-CG-2	180	.9990	6.34	132	.9987	4.67	133	.9986	4.68
P-RS-CG-3	92	.9976	3.70	127	.9986	5.06	130	.9986	5.14
P-RS-CG-4	82	.9970	3.50	124	.9985	5.24	130	.9986	5.44
Ratios									
P-CG-J/P-RS-CG-J		$J=2$	1.71			1.75			1.65
		$J=3$	1.65			1.60			1.61

TABLE 4
CYBER 205 *run time* (*sec*) *for various methods applied to the modified Wachspress problem for group* 1.

Uniform mesh	90°			60°			30°		
Method	Iter	M_E	Time	Iter	M_E	Time	Iter	M_E	Time
CG	1366	.9999	31.64	1375	.9999	31.83	2000+	.9999	
PT-CG	1034	.9999	23.36	1033	.9999	23.34	1694	.9999	38.24
P-CG-1	355	.9991	16.93	432	.9993	20.16	767	.9980	36.53
P-CG-2	184	.9978	10.24	248	.9987	13.79	595	.9997	32.96
P-CG-3	152	.9972	9.09	216	.9985	12.89	583	.9997	34.81
CCSI-H	496	.9997	9.70	551	.9998	10.77	1019	.9999	19.91
RS-CG	358	.9995	8.56	442	.9996	10.57	781	.9934	18.65
P-RS-CG-2	181	.9978	5.85	248	.9987	7.90	595	.9997	18.84
P-RS-CG-3	153	.9972	5.53	216	.9985	7.76	583	.9997	20.78
P-RS-CG-4	151	.9971	5.60	221	.9985	8.41	583	.9997	22.68
Ratios									
P-CG-J/P-RS-CG-J		$J=2$	1.75			1.75			1.75
		$J=3$	1.64			1.66			1.68



$\Delta X = \Delta S = 0.125$ cm

$D_1 = 1.4$, $\sigma_1 = .04$, $S_1 = 1.0$, $D_2 = 1.5$, $\sigma_2 = .08$, $S_2 = 0.0$
$D_3 = 3.0$, $\sigma_3 = .01$, $S_3 = 0.0$ (see Ref. 1).
FIG. 1. *Modified Wachspress problem.*

methods applied to the original block tridiagonal systems. In practice, the improvement possible using the reduced system methods will be a function of the problem, the number of incomplete cyclic reduction steps applied, and the computer under consideration.

(ii) Numerical results demonstrate that the most effective methods are the P-RS-CG-J reduced system conjugate gradient method with preconditioner based on incomplete block odd-even cyclic reduction and the RS-CG reduced system conjugate gradient method with preconditioner D_B. Of these methods, P-RS-CG-J is best for problems with strong coupling in the y (traverse) direction.

(iii) Preconditioning based on incomplete block odd-even cyclic reduction is very effective for reducing computational cost for most of the problems considered.

(iv) The DXY program solves two-dimensional problems that fit in the CYBER 205 central memory. For larger two-dimensional problems or for three-dimensional problems, the cyclic Chebyshev and reduced system Chebyshev methods are expected to be more competitive because of their ability to perform concurrent iterations [12], reducing data transfer costs. For these methods, iterations $n+1, n+2, \cdots, n+k$ for some k can be initiated while iteration n is in progress and k iterations can be completed during a single sweep of the data through central memory [17].

(v) As expected [10], the results obtained for the cyclic Chebyshev method CCSI-H applied to the red/black system (3) and (4), and the reduced system Chebyshev method RS-SI applied to (5) corresponding to the choice of preconditioner D_B, are essentially identical. These methods are competitive with other preconditioned methods applied to block tridiagonal systems.

Finally, the main advantage of preconditioning is realized when the spectrum of eigenvalues of the iteration matrices is constricted. For convenience, $m_E=-1.0$ was used as an initial estimate of the algebraically smallest eigenvalue of the iteration matrices for the P-SI-J and P-RS-SI-J entries reported in Table 1 corresponding to the Chebyshev method. The CCSI-H and RS-SI methods do not require an estimate m_E [12]. The main point here is that the choice of acceleration parameters for the Chebyshev method can always be fine-tuned for special classes of problems and this leads to improved convergence and reduced computational cost.

Acknowledgments. I wish to thank L. A. Hageman for stimulating discussions on this work during the past four years; R. B. Smith and C. A. Harrison for useful suggestions; and the referees for many helpful comments and for pointing out the papers of Axelsson, Brinkkemper, and Il'in [3], [4] and Elman [7]. Thanks are also due Joan J. Nichol and her colleagues in the Word Processing Department for typing various drafts of this paper.

REFERENCES

[1] I. K. ABU-SHUMAYS, *Comparison of methods and algorithms for tridiagonal systems and for vectorization of diffusion computions*, in Supercomputer Applications, R. W. Numrich, ed., Plenum Press, New York, London, 1985, pp. 29-56 (see also *Vectorization of transport and diffusion computations on the* CDC CYBER 205, Nucl. Sci. Engrg., 92 (1986), pp. 4-19).

[2] ———, *Preconditioned polynomial iterative acceleration methods for block tridiagonal systems*, Report WAPD-T-2885, Bettis Atomic Power Laboratory, September 2, 1986.

[3] O. AXELSSON, *A survey of vectorizable preconditioning method for large scale finite element matrix problems*, Center for Numerical Analysis Report CNA-190, University of Texas, Austin, TX, February 1984.

[4] O. AXELSSON, S. BRINKKEMPER, AND V. P. IL'IN, *On some versions of incomplete block-matrix factorization iterative methods*, Linear Algebra Appl., 58 (1984), pp. 3-15.

[5] 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.

[6] T. DUPONT, R. P. KENDALL, AND H. H. RACHFORD, JR., *An approximate factorization procedure for solving self-adjoint elliptic difference equations*, SIAM J. Numer. Anal., 5 (1968), pp. 559-573.

[7] H. C. ELMAN, *Approximate Schur complement preconditioners on serial and parallel computers*, Revision of Tech. Report 1704, Computer Science Department and Institute for Advanced Computer Studies, University of Maryland, College Park, MD, August 1987.

[8] D. J. EVANS, *Preconditioning Methods: Analysis and Applications*, Topics in Comput. Math., Vol. 1, Gordon and Breach, New York, 1983.

[9] L. A. HAGEMAN, *Block iterative methods for two-cyclic matrix equations with special application to the numerical solution of the second-order self-adjoint elliptic partial differential equations in two dimensions*, Report WAPD-TM-327, Bettis Atomic Power Laboratory, 1962.

[10] L. A. HAGEMAN, F. T. LUK, AND D. M. YOUNG, *On the equivalence of certain iterative acceleration methods*, SIAM J. Numer. Anal., 17 (1980), pp. 852-873.

[11] L. A. HAGEMAN AND R. S. VARGA, *Block iterative methods for cyclically reduced matrix equations*, Numer. Math., 6 (1964), pp. 106-119.

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

[13] D. HELLER, *Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems*, SIAM J. Numer. Anal., 13 (1976), pp. 484-496.

[14] D. S. KERSHAW, *The solution of single linear tridiagonal systems and vectorization of the* ICCG *Algorithm on the* CRAY 1, Report UCID-19085, Lawrence Livermore Laboratory, Livermore, CA, June 25, 1981, also in *Parallel Computations*, G. Rodrigue, ed., Academic Press, New York, 1982, pp. 85-100.

[15] J. A. MEIJERINK AND H. A. VAN DER VORST, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148-162.

[16] G. MEURANT, *The block preconditioned conjugate gradient method on vector computers*, BIT, 24 (1984), pp. 623-633.

[17] C. J. PFEIFER AND C. J. SPITZ, PDQ-08 *reference manual*, Report WAPD-TM-1266, Bettis Atomic Power Laboratory, May 1978.

[18] G. RODRIGUE AND D. WOLITZER, *Preconditioning by incomplete block cyclic reduction*, Math. Comp., 42 (1984), pp. 549-565.

[19] R. S. VARGA, *Factorization and normalized iterative methods*, in Boundary Problems in Differential Equations, R. E. Langer, ed., University of Wisconsin Press, Madison, WI, 1960, pp. 121-142.

[20] ———, *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1962.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 3, pp. 563-592, May 1990

OPTIMIZING TRIDIAGONAL SOLVERS FOR ALTERNATING DIRECTION METHODS ON BOOLEAN CUBE MULTIPROCESSORS*

CHING-TIEN HO† AND S. LENNART JOHNSSON‡

Abstract. Sets of tridiagonal systems occur in many applications. Fast Poisson solvers and Alternate Direction Methods make use of tridiagonal system solvers. Network-based multiprocessors provide a cost-effective alternative to traditional supercomputer architectures. The complexity of concurrent algorithms for the solution of multiple tridiagonal systems on Boolean-cube-configured multiprocessors with distributed memory are investigated. Variations of odd-even cyclic reduction, parallel cyclic reduction, and algorithms making use of data transposition with or without substructuring and local elimination, or pipelined elimination, are considered. A simple performance model is used for algorithm comparison, and the validity of the model is verified on an Intel iPSC/1. For many combinations of machine and system parameters, pipelined elimination, or equation transposition with or without substructuring is optimum. Hybrid algorithms that at any stage choose the best algorithm among the considered ones for the remainder of the problem are presented.

It is shown that the optimum partitioning of a set of independent tridiagonal systems among a set of processors yields the embarrassingly parallel case. If the systems originate from a lattice and solutions are computed in alternating directions, then to first order the aspect ratio of a computational lattice shall be the same as that of the lattice forming the base for the equations.

The experiments presented here demonstrate the importance of combining in the communication system for architectures with a relatively high communications start-up time.

Key words. tridiagonal systems, Boolean cubes, pipelined Gaussian elimination, transposition, substructuring, balanced cyclic reduction

AMS(MOS) subject classifications. 15A06, 65F05, 65N05, 68Q25

1. Introduction.

Tridiagonal systems of equations occur in many methods used for the solution of partial differential equations. In the Alternating Direction Method (ADM), tridiagonal matrices arise from one-directional central difference approximations of partial derivatives. In so-called fast Poisson solvers a Fast Fourier Transform (FFT) in one dimension decouples the system into a number of independent tridiagonal systems. Tridiagonal systems are also used as preconditioners for the conjugate gradient method.

In parallel architectures with distributed memory, the communication is a critical issue with respect to performance, in particular, if the number of processors approaches the number of unknowns [25], [23], [24], [3], [11], [21]. The communication time for a given computation on a given architecture depends on the data allocation to the distributed memory, the choice of numerical algorithm, and the communication algorithms. These issues are the focal points of this work. The target architectures have processors interconnected as a Boolean n-cube, which has $N=2^n$ nodes. One- and multidimensional lattices can be embedded in Boolean cubes with one array node per cube node preserving adjacency by a *binary-reflected* Gray code [22], [18]. This embedding uses all cube nodes when the lattice sides are powers of two. For other

* Received by the editors July 31, 1987; accepted for publication (in revised form) June 23, 1989. This work was supported by the Office of Naval Research under contract N00014-86-K-0310.

† IBM Almaden Research Center, K54/802, 650 Harry Road, San Jose, California 95120 (ho@ibm.com). This work was done while the author was at the Department of Computer Science, Yale University, New Haven, Connecticut.

‡ Departments of Computer Science and Electrical Engineering, Yale University, New Haven, Connecticut 06520, and Thinking Machines Corporation, Cambridge, Massachusetts 02142 (Johnsson @ cs. Yale. edu; Johnsson @ think. com).

side lengths, other embeddings may be more effective [7], [2]. In our analysis the time for the communication of one byte between a pair of processors is denoted t_c, the time for the communication overhead (start-up) is denoted τ, the time for an arithmetic operation t_a, and the maximum packet size imposed by a limited buffer size B_m. Moreover, we assume that communication can take place only on one port per processor at a time, *one-port communication.* The time for the solution of the systems of equations is denoted $T_{\text{alg}}(P, Q; N, t_a, t_c, \tau, B_m)$, where alg identifies the algorithm, the first argument is the size (P) of a tridiagonal system and the second the number of systems (Q). The third parameter is the number of Boolean cube processors to which the $P \times Q$ equations are allocated. When necessary, other parameters may be added or omitted.

For the solution of a single tridiagonal system we consider:

- Substructuring followed by odd-even cyclic reduction, CR, [1] for the reduced system, as described in [18] and [10].
- Substructuring followed by parallel cyclic reduction, PCR [9].
- The gathering of the equations to 2^{n-i} processors by i steps of a data transposition algorithm such that each processor holds $(P/N)2^i$ equations, followed by two-way elimination, and a final phase in which the results are distributed to the set of processors originally holding the equations; algorithm TGET (i) for i-step data gathering by *Transposition, Gaussian elimination,* and i-step data scattering by *Transposition.* With substructuring algorithm TGET (i) becomes algorithm SS/TGET (i).
- The broadcasting of equations from every node to every other node within i-dimensional subcubes such that the system is evenly distributed across 2^{n-i} processors and there are 2^i copies of the same system. No distribution of the solution is needed between the $(n-i)$-dimensional subcubes. With substructuring, algorithm BCGE (i) becomes algorithm SS/BCGE (i).
- A set of hybrid algorithms obtained by combining substructuring, odd-even cyclic reduction, and the TGET (i) algorithm.

The first transpose operation in TGET (i) is a gather operation, and the second a scatter operation. We refer to both as transpose operations since each gather, or scatter, operation is equivalent to a vector transpose. Furthermore, in the case of multiple tridiagonal systems the multiple gather, or scatter, operations are transpose operations. In the TGET (i) algorithm only two processors are used for the elimination. The purpose of the transpose operation is to reduce the number of communication start-ups compared to a direct two-way elimination. The reduction is accomplished by using a transpose algorithm that requires i steps in moving the equations to an $(n-i)$-cube. With *one-port communication* a doubling algorithm such as a spanning *binomial tree* algorithm is optimal [15]. The data transfer time increases for every step of the transpose algorithm, and is proportional to $P/N(2^i-1)$ for P equations and a spanning binomial tree algorithm. In the case of concurrent communication on all ports of every processor, *n-port communication,* the data-transfer time can be reduced by a factor of i by using communication according to *balanced trees* [15], [8]. Depending on the ratio of the data-transfer time to the start-up time there may exist a nontrivial, optimum value of i.

Cyclic reduction algorithms make use of many processors for the elimination, and yield a lower, parallel, arithmetic complexity for $N \geqq 16$ (see Table 1). But the lower bound for the number of start-ups for a vector transposition is approximately half of the minimum number of start-ups for odd-even cyclic reduction with communications restricted to one send *or* one receive operation at a time. A lower bound of n start-ups for the transposition is shown in [17]. Although the number of start-ups for cyclic reduction is higher in the case of unlimited buffer sizes, it has a lower data-transfer time than the transpose algorithm, and possibly also fewer start-ups in the case of buffers of limited size. These different characteristics give rise to some interesting optimization problems that we investigate in the context of hybrid algorithms.

For the solution of multiple tridiagonal systems we consider:

- Pipelined two-way Gaussian elimination.
- Gathering of equations and separation of systems into subcubes by equation transposition, solution of systems by pipelined two-way Gaussian elimination, and scattering of the solution. This algorithm is the multiple systems version of algorithms TGET (i). With initial substructuring, it becomes SS/TGET (i).
- Substructuring followed by *Balanced Cyclic Reduction* [14] for the reduced system solver, SS/BCR.
- Combinations of the above three algorithms.

We have verified our analytical complexity models on a medium scale, parallel architecture: the Intel iPSC/1. A binary-reflected Gray code was used for the data mapping. The Intel iPSC/1 effectively only supports one send *or* one receive operation per processor at a time, and most of our complexity estimates are specialized to this case. Because of the high copying time on the Intel iPSC/1 there exists a packet size $B_{\text{cp}} < B_m$ above which it is beneficial for sending data directly rather than combining several packets into one, in order to minimize the number of start-ups [17]. For the system software available at the time of the experiments $B_{\text{cp}} = 256 \approx \tau / t_{\text{cp}}$ bytes, where the time for copying one byte is t_{cp}. The copy time is sufficiently large to affect the choice of algorithm. The start-up time for communication is also significant on the Intel iPSC/1, and we demonstrate the effectiveness of message combining to reduce communications overhead.

2. Solving a single tridiagonal system on an n-cube. We assume that the equations are assigned to processors by dividing the system into blocks, or by applying incomplete nested dissection [5] and suitably associating separator nodes with adjacent partitions. The partitioned tridiagonal system of equations has the following form:

$$\left(\begin{array}{cccccccccccc}
x & x & & & & & & & & & & \\
x & x & x & & & & & & & & & \\
 & x & x & x & & & & & & & & \\
\hline
 & & x & x & x & & & & & & & \\
 & & & x & x & x & & & & & & \\
 & & & & x & x & x & & & & & \\
\hline
 & & & & & x & x & x & & & & \\
 & & & & & & x & x & x & & & \\
 & & & & & & & x & x & x & & \\
\hline
 & & & & & & & & x & x & x & \\
 & & & & & & & & & x & x & x \\
 & & & & & & & & & & x & x
\end{array}\right) X = Y.$$

The horizontal lines indicate the partitioning/substructuring. For each partition we apply Gauss-Jordan elimination as described in [10] and [26]. One communication is needed for the first and the last equations in each partition. By a suitable allocation of matrix coefficients to processors it is not necessary to transfer an entire equation, and some element transfers [19] can be saved. The execution time is approximately

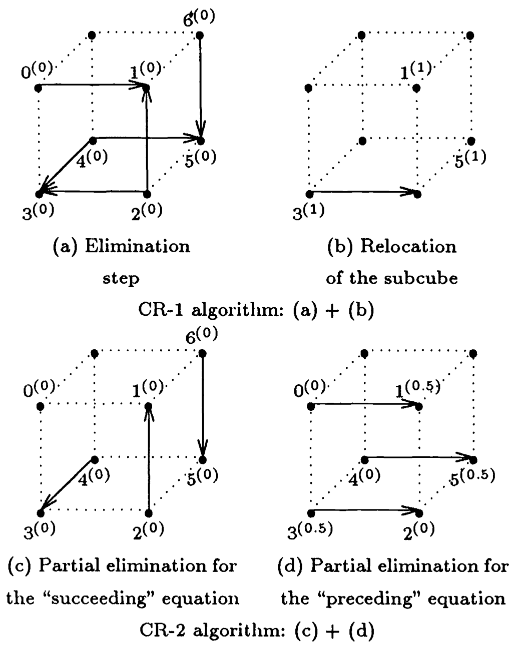
$$(1)\quad T_{SS}(P, 1; N, t_a, t_c, \tau) = 17\left(\left\lceil \frac{P}{N} \right\rceil - 1\right) t_a + 2(16t_c + \tau) + 2(4t_c + \tau), \qquad B_m \geqq 16,$$

where it is assumed that a processor can perform one send *or* one receive operation at a time, that the system is real, and that floating-point numbers are represented by four bytes (single-precision). The form of the system after the substructured elimination is shown below. Note that the elimination of the last block is done in reverse order such that the reduced system formed by the last equation of each block, but the last, is of order $2^n - 1$.

$$\left(\begin{array}{cccccccccccc}
x & & x & & & & & & & & & \\
 & x & x & & & & & & & & & \\
 & & x & & & x & & & & & & \\
\hline
 & & x & x & & x & & & & & & \\
 & & x & & x & x & & & & & & \\
 & & x & & & x & & & x & & & \\
\hline
 & & & & & x & x & & x & & & \\
 & & & & & x & & x & x & & & \\
 & & & & & x & & & x & & & \\
\hline
 & & & & & & & & x & x & & \\
 & & & & & & & & x & & x & \\
 & & & & & & & & x & & & x
\end{array}\right) X = Y.$$

2.1. Cyclic Reduction (CR). Odd-even cyclic reduction requires 17 operations per unknown, the same as substructured Gaussian elimination. Substructuring is always preferable when using odd-even cyclic reduction, since only four communications are needed for the substructuring with one send *or* one receive at a time.

In [10] and [14] an *exchange* algorithm, and an *in-place* algorithm are proposed for the solution of tridiagonal systems distributed with one equation per processor in a Boolean n-cube. In the *exchange* algorithm the active set of equations are recursively moved into an easily identified subcube of one less dimension for each elimination step. Communication is needed for both the elimination step (Fig. 1(a)) and the reallocation to the subcubes (Fig. 1(b)). The reallocation can be performed with only one communication, such that with the communication restricted to one send *or* receive operation at a time, $3(n-1)$ communications suffice for the elimination *and* the reallocation. The same number of communications is required in the back substitution phase. We refer to this algorithm as SS/CR-1. Note that in Fig. 1(a), the processor holding equation $3^{(0)}$ needs to store the equation received from the preceding processor ($2^{(0)}$) to avoid using $O(n)$ memory per processor. The number of communications per step can be reduced from three to two at the expense of $O(n)$ memory per processor, instead of constant memory. By performing a partial elimination (Fig. 1(c)), then moving the equation subject to elimination to the processor for the next elimination step, and completing the elimination there (Fig. 1(d)), one communication is saved. A total of $4(n-1)$ communications are needed. This algorithm is called SS/CR-2. For the *in-place* algorithm, SS/CR-IR, we simply use the routing software of the Intel iPSC.



(a) Elimination step (b) Relocation of the subcube

CR-1 algorithm: (a) + (b)

(c) Partial elimination for the "succeeding" equation (d) Partial elimination for the "preceding" equation

CR-2 algorithm: (c) + (d)

FIG. 1. *One forward elimination step of* CR-1 *and* CR-2 *algorithms on a* 3*-cube.*

The time to solve a single tridiagonal system of $P=2^n-1$ real equations on an n-cube by the SS/CR-2 algorithm [10], [14] is given by

$$(2)\quad \begin{aligned} T_{\text{CR-2}}(N-1,1;\,N,t_a,t_c,\tau) &= (\log N-2)(16t_a+2(16+4)t_c+2\times 2\tau) \\ &\quad +(14+3)t_a+2(16+4)t_c+2\times 2\tau, \qquad B_m \geqq 16. \end{aligned}$$

For $P>N$ substructuring is applied and the estimated time is given by

$$(3)\quad T_{\text{SS/CR-2}}(P,1;\,N,t_a,t_c,\tau)=T_{\text{SS}}(P,1;\,N,t_a,t_c,\tau)+T_{\text{CR-2}}(N-1,1;\,N,t_a,t_c,\tau).$$

In equation (2) we assume that the division normally carried out in the back substitution phase is performed concurrently for all equations after the elimination phase. With real coefficients, one right-hand side, and one processor per equation, a time of $3t_a$ is required. The arithmetic complexity can be reduced to 11 [14] sequential operations per step (instead of 16) without an increase in the communication complexity by dividing the arithmetic operations for the elimination and backsubstitution phases among pairs of processors. A further reduction to nine operations is possible at the expense of additional communication. For complex coefficients the number of arithmetic operations per step is 82, 47, and 43, respectively [13]. The techniques for lower parallel arithmetic complexity are not applicable to the case of multiple tridiagonal systems. The complexity estimates in this paper apply to algorithms using 16 arithmetic operations in sequence for each equation, and a compatible communication scheme. The predicted and measured times agree well for the Intel iPSC/1 [16].

2.2. Parallel Cyclic Reduction (PCR). Parallel Cyclic Reduction [9] performs an elimination on every equation in every step. With one equation per processor all processors are active throughout the computation, which completes in n steps. After each step the problem is transformed such that the number of equations per tridiagonal system is halved, and the number of systems doubled:

$$\begin{pmatrix} x & x & & & & & & \\ x & x & x & & & & & \\ & x & x & x & & & & \\ & & x & x & x & & & \\ & & & x & x & x & & \\ & & & & x & x & x & \\ & & & & & x & x & x \\ & & & & & & x & x \end{pmatrix} \Rightarrow \begin{pmatrix} x & & x & & & & & \\ & x & & x & & & & \\ x & & x & & x & & & \\ & x & & x & & x & & \\ & & x & & x & & x & \\ & & & x & & x & & x \\ & & & & x & & x & \\ & & & & & x & & x \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} x & & & & x & & & \\ & x & & & & x & & \\ & & x & & & & x & \\ & & & x & & & & x \\ x & & & & x & & & \\ & x & & & & x & & \\ & & x & & & & x & \\ & & & x & & & & x \end{pmatrix}.$$

No backsubstitution is needed. The total arithmetic complexity of the PCR algorithm is $12n2^n$, the parallel complexity $12n$. Substructuring is always preferable. The total number of messages is $2n2^n - 2n + 1$, and the number of parallel communications in sequence $4n$ (two exchanges) with one send *or* one receive operation at a time. During step j, $0 \leqq j < n$, the ith row of the reduced system exchanges its data with the $(i-2^j)$th and the $(i+2^j)$th rows (if $i-2^j \geqq 0$, and $i+2^j \leqq P-1$). With a binary-reflected Gray code encoding i and $i+2^j$ are at most a distance two apart in the cube [10]. In implementing the SS/PCR algorithm on a Boolean cube, we can:

(1) Perform an exchange operation after each step of PCR to move each subset of equations into the same subcube, such that communications for the next step remains *nearest-neighbor* for each subset, with no interference between communication for different subsets. This is algorithm SS/PCR-1.

(2) Combine exchange communications with communications for elimination, as in the SS/CR-2 algorithm, and arrive at a SS/PCR-2 algorithm.

(3) Decompose and combine the two distance-two sends and receives into three exchanges, i.e., six "nearest-neighbor" sends or receives. This is algorithm SS/PCR-3.

(4) Use a static allocation and the routing logic, SS/PCR-IR.

In SS/PCR-1, $6n-2$ start-ups are required. In the SS/PCR-2 algorithm the number of start-ups is reduced to $4n$ by splitting the elimination on the row into two parts; one equation is received and the associated elimination performed, then the partially modified row is sent to the "after-exchanged" processor, and the elimination completed using the row in the new processor. The allocation of equations to processors is changed. The solution variables are allocated according to binary code encoding of the indices. To move the solution of the reduced system back to the corresponding partition, a binary code to Gray code conversion is required. Such a conversion requires $2n-2$ start-ups [12]. The total number of start-ups is $8n-4$ for SS/PCR-1 and $6n-2$ for SS/PCR-2. In the SS/PCR-3 algorithm, the distance-two sends or receives are decomposed into two "nearest-neighbor" communications performed concurrently for all nodes such that there is no edge conflict [12]. Moreover, since the data sent to both distance-two neighbors in each SS/PCR step are the same, and the two paths to the distance-two neighbors can be arranged such that they share the first edge, the number of start-ups can be reduced from $8n-4$ to $6n-2$, i.e., the same as for the SS/PCR-2 algorithm. Note that the SS/PCR-3 algorithm is an *in-place* algorithm and therefore does not require the binary code to Gray code conversion at the end. In the SS/PCR-IR algorithm the communication time depends entirely on the routing logic. A routing discipline, such as the one used in the Intel iPSC, that routes the dimensions that need to be routed in increasing order yields a conflict-free routing for the SS/PCR-IR algorithm [10].

For the analysis we use the following estimated time for PCR-2 (and PCR-3):

$$(4)\qquad T_{\text{PCR-2,3}}(N, 1; N, t_a, t_c, \tau) = \log N(12t_a + 6\times 16t_c + 6\tau) - 2(16t_c + \tau), \qquad B_m \geqq 16,$$

and for the substructured version:

$$(5)\quad T_{\text{SS/PCR-2,3}}(P, 1; N, t_a, t_c, \tau) = T_{\text{SS}}(P, 1; N, t_a, t_c, \tau) + T_{\text{PCR-2,3}}(N, 1; N, t_a, t_c, \tau).$$

The predicted times for algorithms PCR-3 and PCR-IR are significantly lower than the measured times, and also significantly higher than the measured times for the odd-even cyclic reduction algorithms [16]. We attribute the difference between the measured and predicted times for the PCR algorithms to synchronization delays. The PCR algorithm is not of interest for multiple systems per processor due to its higher arithmetic complexity compared to odd-even cyclic reduction.

2.3. Equation transposition and Gaussian elimination. Conventional Gaussian elimination only requires eight operations per unknown, but substructured elimination requires 17 operations per unknown. Two-way elimination allows two processors to be used for the same tridiagonal system (almost) without an increase in total arithmetic complexity. With respect to arithmetic complexity substructuring shall be used if $N > 17/4$. Note that the start-up time of TGET $(n-2)$ is the same as that of TGET $(n-1)$ in the event of one start-up per communication (unbounded buffers). When the equations are gathered by a doubling algorithm, such as a spanning binomial tree algorithm [6], the number of communication start-ups is reduced compared to a direct two-way elimination for a sufficiently large buffer size. After gathering the equations by an i-step transpose algorithm the equations are allocated in an $(n-i)$-dimensional subcube with $(P/N)2^i$ equations per processor, if no substructuring was applied initially; otherwise there are 2^i equations per processor. After the i steps of equation gathering the set of equations can either be solved by two-way elimination, or by substructuring followed by the solution of a system with $2^{n-i}-1$ equations distributed with one equation per processor in an $(n-i)$-cube.

The time for i steps of equation gathering and i steps of scattering of the solution from an $(n-i)$-cube to an n-cube is

$$(6)\qquad T_{\text{sbt}}(P, 1; N, t_c, \tau, B_m, i) = \sum_{j=0}^{i-1}\left(\left\lceil \frac{16P\times 2^j}{B_m N}\right\rceil + \left\lceil \frac{4P\times 2^j}{B_m N}\right\rceil\right)\tau + 20\frac{P}{N}(2^i-1)t_c.$$

We refer to the algorithm that performs "i steps of gathering, two-way elimination for the solution of the systems of equations, and i-steps of scattering of the solutions" as a TGET (i) algorithm. However, for an optimum combination of substructuring and Gaussian elimination the choice of continuing the gathering of equations, or solving the system by two-way elimination, or performing a substructuring operation should be reevaluated for each step. We consider such algorithms in the section about hybrid algorithms. Let $T_{2GE}(P, 1; N, t_a, t_c, \tau)$ be the time for two-way Gaussian elimination performed on P equations distributed across $N > 1$ processors. Then,

$$T_{2GE}(P, 1; N, t_a, t_c, \tau) = (4P-2)t_a + N\tau + (8N+8)t_c. \tag{7}$$

In this expression it is assumed that both processors involved in the elimination of the "middle" 2×2 system solve for one variable after an exchange of the coupling equations. It is possible to save two element transfers at the expense of an increased arithmetic complexity of two operations by having one processor send one equation to the other, which on the solution of a 2×2 system returns one of the solution variables.

For $P > N = 4$, $T_{2GE}(P, 1; 4, t_a, t_c, \tau) < T_{SS}(P, 1; 4, t_a, t_c, \tau) + T_{2GE}(4, 1; 4, t_a, t_c, \tau)$. Hence, with P equations spread across four processors, substructuring should never be applied. Two-way Gaussian elimination has a processor efficiency of 50 percent with eight floating-point operations per equation, whereas substructuring has 100 percent efficiency, but requires 17 floating-point operations per equation. Furthermore, two-way Gaussian elimination has the same communication time regardless of the number of equations per processor. Substructuring needs an additional time of $4\tau + 40t_c$.

The gathering of equations to a subset of processors, *all-to-one personalized communication* [15] in subcubes, can either be carried out by a send operation using the general router of the Intel iPSC/1, or by a user coded spanning binomial tree algorithm with combining (which is optimal for one-port communication [6]). The router also uses a binomial tree routing, but does not perform message combining. For the scatter operation, *one-to-all personalized communication*, there is the additional possibility of a *one-to-all broadcasting* of the entire solution vector. More data than necessary is communicated, no splitting of packets is required, and many architectures have efficient implementations of data broadcast. In order to evaluate the different algorithms for gathering and scattering equations on the Intel iPSC/1, we implemented:

- *All-to-one personalized communication* using the router (no combining) followed by router broadcast.
- *All-to-one personalized communication* with combining and *one-to-all personalized communication* with splitting, both based on the spanning binomial tree routing [15].
- *All-to-one personalized communication* with combining followed by router broadcast.

We implemented three different communication algorithms for the SS/TGET (n) algorithm. The measured times compare as shown in Fig. 2. The substructuring part is the same for all alternatives and is not included. We refer to the three communication algorithms with the postfix -sb, -cc, and -cb. The measured times for the SS/TGET (n)-sb algorithm increase exponentially due to the lack of message combining. The implementation using combining and broadcasting is more efficient than the implementation using combining for both phases of the transposition. The latter is about 10 percent slower than the former. The arithmetic times are less than 50 percent of the total times for cubes of less than five dimensions. Since the number of communications in the case of an unlimited buffer size increases linearly with the number of dimensions of the cube, but the data volume and arithmetic time increases exponentially, the time for data transfer and arithmetic will eventually dominate the total time as the number of cube dimensions increases. With a limited package size, the number of start-ups will eventually also increase exponentially in the number of cube dimensions.

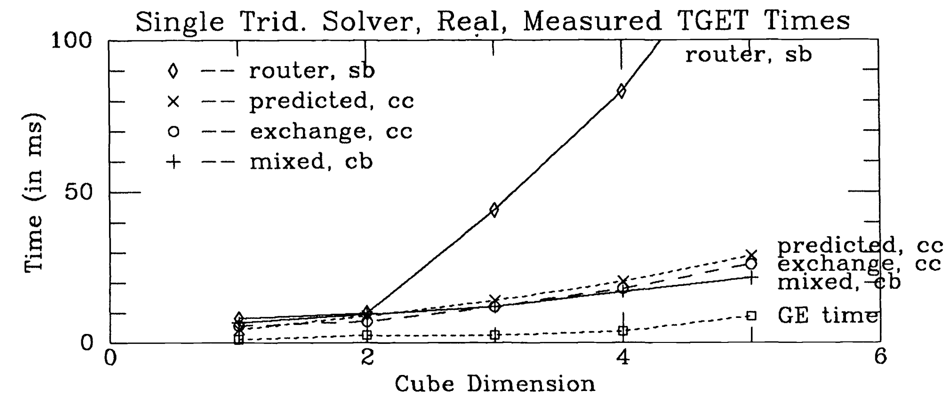


FIG. 2. *Measured values of* $T_{\mathrm{TGET}(n)}(N, 1;\ N, t_a, t_c, \tau)$ *using routing with combining* (-cc), *the routing logic of the* Intel iPSC/1 (*no combining*), *and broadcasting* (-sb), *and combining and broadcasting* (-cb).

2.4. Broadcasting and concurrent two-way Gaussian elimination. In the TGET (i) algorithm, only one $(n-i)$-dimensional subcube is active. It is possible to avoid distributing the results of the two-way elimination to the processors storing the equations, if all processors send their equation(s) to every other processor, *all-to-all broadcasting* [15]. If the broadcasting from every processor is restricted to an i-cube, then the original problem is transformed into solving 2^i identical systems in 2^i independent $(n-i)$-cubes. In each subcube it is sufficient to solve for 2^{n-i} variables. No communication of the components of the solution vector between subcubes is necessary. (If $i=n$, then backsubstitution can be avoided entirely by making the forward elimination terminate at the appropriate equation.) The algorithm can be used with or without a substructuring phase, called SS/BCGE (i) and BCGE (i), respectively. With one send *or* one receive operation at a time, the communication complexity of the broadcasting phase of the BCGE (i) algorithm is twice that of the gathering of equations in algorithm TGET (i). The parallel arithmetic complexity of algorithm BCGE (i) is the same as that of TGET (i). The estimated execution times without any intermediate substructuring and two-way elimination for the equations are

$$(8)\quad \begin{aligned} T_{\mathrm{BCGE}(i)}(P, 1;\ N, t_a, t_c, \tau, B_m, i) &= \left(32\frac{P}{N}(2^i-1)+2^{n-i+3}+8\right)t_c \qquad B_m \geqq 12 \\ &\quad + 2\left(\sum_{j=0}^{i-1}\left\lceil\frac{16P\times 2^j}{NB_m}\right\rceil+2^{n-i-1}\right)\tau+(4P-2)t_a, \end{aligned}$$

$$(9)\quad \begin{aligned} &T_{\mathrm{SS/BCGE}(i)}(P, 1;\ N, t_a, t_c, \tau, B_m, i) \\ &\quad = T_{\mathrm{SS}}(P, 1;\ N, t_a, t_c, \tau)+T_{\mathrm{BCGE}(i)}(N, 1;\ N, t_a, t_c, \tau, B_m, i). \end{aligned}$$

The measured times are somewhat lower than the predicted times [16] for algorithm BCGE $(n-1)$ assuming one send *or* receive operation at a time. A 20 percent overlap between send and receive operations explains the difference between the predicted and measured times. Such an overlap has also been observed in other experiments. Note that if the communication time for one exchange operation is twice that of one send or one receive operation, then algorithm BCGE (i) is always inferior to algorithm TGET (i). However, if send and receive operations can be performed concurrently, then the BCGE (i) algorithm is of lower complexity than the TGET (i) algorithm.

The BCGE (i) algorithm is of sequential complexity $O(2^iP)$. With multiple tridiagonal systems distributed with one equation per processor, and each processor having one equation of every system, the arithmetic complexity of algorithm BCGE (i) makes it clearly inferior to algorithm TGET (i).

2.5. Hybrid algorithms. The parallel arithmetic complexity of algorithm TGET (i) is higher than that of the CR algorithms, while the former requires fewer start-ups than the latter algorithm in the case of unbounded communication buffers. The data transfer times for the CR algorithms are lower than the data transfer time for algorithm TGET (i). These characteristics motivate the consideration of hybrid algorithms. Let $T_{\text{trid}}(P, 1; N, t_a, t_c, \tau, B_m)$ be the total time required to solve a single system of P equations in an N processor cube for given t_a, t_c, τ, and B_m. Assume that P is a power of two, $P \geqq N \geqq 4$ and $B_m \geqq 16$. Then,

$$T_{\text{trid}}(P, 1; N, t_a, t_c, \tau, B_m)$$
$$\leqq \begin{cases} T_{2\text{GE}}(P, 1; 4, t_a, t_c, \tau) \quad \text{if } N = 4, \\ \min\Big\{ T_{2\text{GE}}(N, 1; N, t_a, t_c, \tau), \\ \qquad T_{\text{sbt}}(N, 1; N, t_c, \tau, B_m) + T_{\text{trid}}\left(N, 1; \frac{N}{2}, t_a, t_c, \tau, B_m\right), \\ \qquad\qquad T_{\text{cr}}(t_a, t_c, \tau) + T_{\text{trid}}\left(\frac{N}{2}, 1; \frac{N}{2}, t_a, t_c, \tau, B_m\right)\Big\} \quad \text{if } P = N, \\ \min\Big\{ T_{2\text{GE}}(P, 1; N, t_a, t_c, \tau), \\ \qquad T_{\text{sbt}}(P, 1; N, t_c, \tau, B_m) + T_{\text{trid}}\left(P, 1; \frac{N}{2}, t_a, t_c, \tau, B_m\right), \\ \qquad\qquad T_{\text{SS}}(P, 1; N, t_a, t_c, \tau) + T_{\text{trid}}(N, 1; N, t_a, t_c, \tau, B_m)\Big\} \quad \text{otherwise,} \end{cases}$$

where

$$T_{2\text{GE}}(P, 1; N, t_a, t_c, \tau) = (4P-2)t_a + N\tau + (8N+8)t_c,$$

$$T_{\text{SS}}(P, 1; N, t_a, t_c, \tau) = 17\left(\frac{P}{N} - 1\right)t_a + 4\tau + 40t_c,$$

$$T_{\text{sbt}}(P, 1; N, t_c, \tau, B_m) = \left(\left\lceil \frac{16P}{B_mN} \right\rceil + \left\lceil \frac{4P}{B_mN} \right\rceil\right)\tau + \frac{20P}{N}t_c, \quad \text{and}$$

$$T_{\text{cr}}(t_a, t_c, \tau) = 16t_a + 4\tau + 40t_c.$$

$T_{\text{sbt}}(P, 1; N, t_c, \tau, B_m)$ is the time for gathering of P/N equations per processor in N processors to $2P/N$ equations per processor in $N/2$ processors, *and* scattering $2P/N$ components of the solution vector in each of $N/2$ processors to P/N components of the solution vector in each of N processors. $T_{\text{cr}}(t_a, t_c, \tau) = T_{\text{CR-2}}(N, 1; N, t_a, t_c, \tau) - T_{\text{CR-2}}(N/2, 1; N/2, t_a, t_c, \tau)$ is the time for one-step cyclic reduction using algorithm CR-2. Note that in an N processor cube, only $n-1$ steps is required for the SS/CR algorithm to solve for a single system as seen from equations (2) and (3). Since the "one-step saving" does not generalize to the multiple systems case, we ignore the possible one-step saving for the CR algorithm. For the solution of

P equations on $N=4$ processors, two-way Gaussian elimination yields the lowest complexity of the algorithms we consider. Hence, $T_{\text{trid}}(P, 1; 4, t_a, t_c, \tau, B_m) = T_{2\text{GE}}(P, 1; 4, t_a, t_c, \tau)$. For N equations on $N>4$ processors we consider:

- Two-way Gaussian elimination.
- One-step gathering of equations followed by the solution of a system of N equations evenly distributed over $N/2$ processors.
- One-step cyclic reduction followed by the solution of a system of $N/2$ equations on $N/2$ processors.

For the solution of $P>N$ equations on $N>4$ processors we consider:

- Two-way Gaussian elimination.
- One-step gathering of equations followed by the solution of a system of P equations on $N/2$ processors.
- Substructuring followed by the solution of a system of N equations on N processors.

Note that partial substructuring, i.e., reduction to more than one equation per processor, is in general not preferable with respect to performance. The "total" and "partial" substructurings require the same number of communication start-ups, and the only disadvantage of substructuring compared to the TGET (i) algorithm is the start-up time for $N>4$. (The time for arithmetic operations for substructuring and for the TGET ($n-1$) algorithm compares as $17(P/N)$ to $4P$, approximately.)

Figure 3 shows the calling sequencies. The label on an edge of the graph is the expression for the time required to traverse that edge, with the arguments for the number of systems (which is one) and the parameters t_a, t_c, τ, and B_m omitted. The argument of T_{sbt} and T_{SS} in the figure is P/N. The direct solution of P equations on N processors is not represented in the figure. The cyclic reduction (or parallel cyclic reduction) algorithm is represented by the edges on the diagonal. For each node on the diagonal there is one equation per processor. Applying the TGET (i) algorithm implies a vertical motion to level $P/N=2^i$. Substructuring implies a horizontal motion to the node on the diagonal. Figures 4 and 5 show the optimal calling sequencies (among the considered algorithms) for various ratios of t_a/t_c and τ/t_c, and B_m (with the maximum required buffer size specified). The "∘" sign denotes the two-way Gaussian elimination algorithm in the figures.

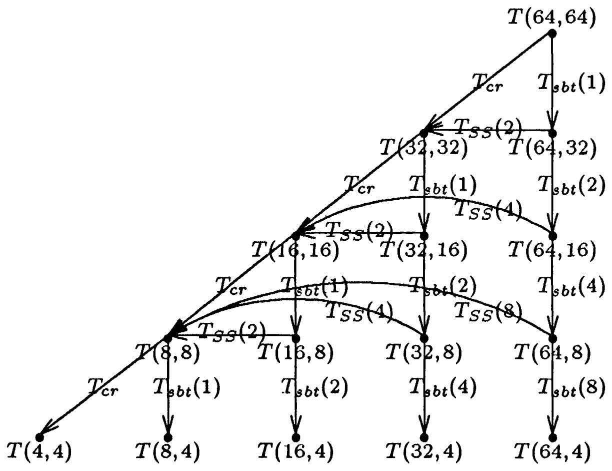


FIG. 3. *The calling sequencies.*

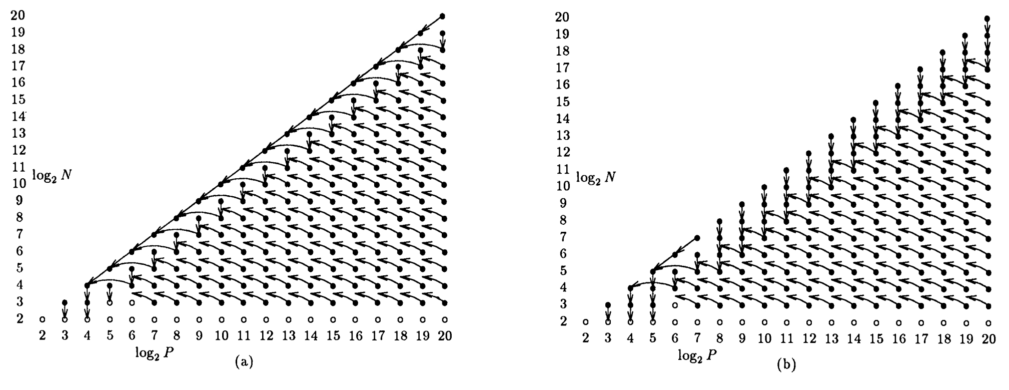
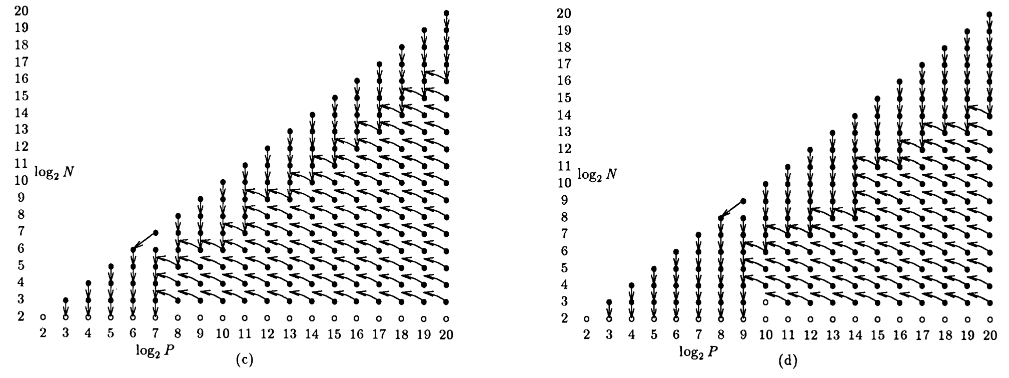



FIG. 4. *The optimal calling sequencies are shown for $t_a/t_c = 10$, $\tau/t_c = 1000$ and* (a) $B_m = 16$ *bytes,* (b) $B_m = 32$ *bytes and* (c) $B_m \geqq 256$ *bytes, respectively. In* (d), *the optimal calling sequencies for $t_a/t_c = 1$, $\tau/t_c = 1000$ and $B_m \geqq 1024$ bytes are shown.*

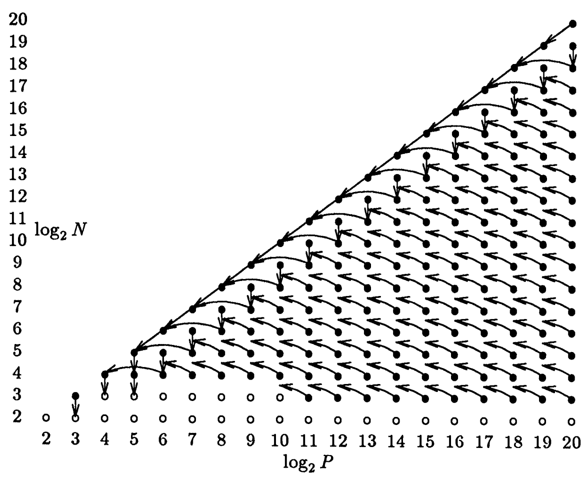


FIG. 5. *The optimal calling sequencies for $t_a/t_c = 0.01$, $\tau/t_c = 1$, and $B_m \geqq 16$ bytes.*

2.6. Summary of algorithms for the solution of single tridiagonal systems. Table 1 summarizes the complexities of the individual algorithms. Memory requirements are expressed in units of four elements of four bytes each (real systems, single precision). Some measured and predicted times for the solution of a single tridiagonal system on an Intel iPSC/1 are given in Fig. 6. The measured values of the machine parameters are approximately: $t_a = 30$ μsec, $t_c = 10$ μsec, and $\tau = 2$ $m$sec. The predicted execution

TABLE 1
Complexity comparison of algorithms for the single tridiagonal system.

Algorithm	Arithmetic	Byte transfers	Min start-ups	Memory
TGET (n)	$8P-7$	$20P\left(1-\frac{1}{N}\right)$	$2n$	P
TGET (i)	$4P-2$	$20\frac{P}{N}(2^i-1)+2^{n-i+3}+8$	$2n+2^{n-i}-2(n-i)$	$\frac{P}{N}2^i$
BCGE (i)	$4P-2$	$32\frac{P}{N}(2^i-1)+2^{n-i+3}+8$	$2n+2^{n-i}-2(n-i)$	$\frac{P}{N}2^i$
SS/CR-1	$17\left\lceil\frac{P}{N}\right\rceil+16n-32$	$60n-40$	$6n-4$	$\frac{P}{N}$
SS/CR-2	$17\left\lceil\frac{P}{N}\right\rceil+16n-32$	$40n$	$4n$	$\frac{P}{N}+\log N$
SS/PCR-2,3	$17\left\lceil\frac{P}{N}\right\rceil+12n-17$	$96n+8$	$6n+2$	$\frac{P}{N}$
SS/TGET (n)	$17\left\lceil\frac{P}{N}\right\rceil+8N-24$	$20N+20$	$2n+4$	$\frac{P}{N}+N$
SS/TGET (i)	$17\left\lceil\frac{P}{N}\right\rceil+4N-19$	$20(2^i-1)+2^{n-i+3}+48$	$2n+2^{n-i}-2(n-i)+4$	$\frac{P}{N}+2^i$
SS/BCGE (i)	$17\left\lceil\frac{P}{N}\right\rceil+4N-19$	$32(2^i-1)+2^{n-i+3}+48$	$2n+2^{n-i}-2(n-i)+4$	$\frac{P}{N}+2^i$

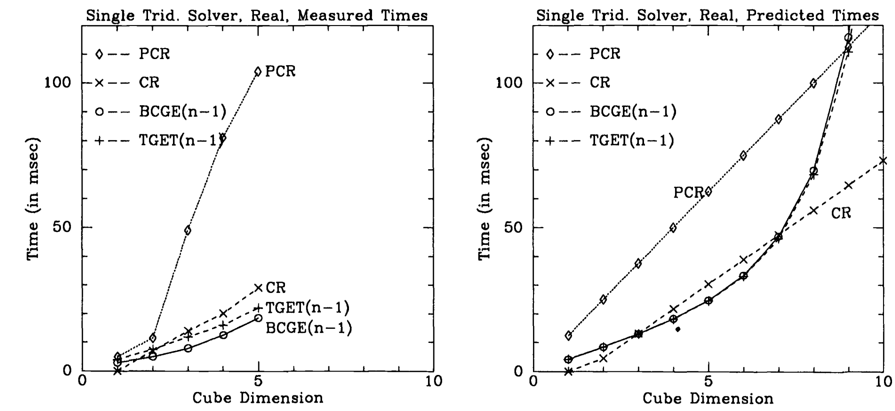


FIG. 6. *Measured and predicted times of the* CR-2, PCR-3, TGET $(n-1)$-cc, *and* BCGE $(n-1)$ *algorithms on the* Intel iPSC/1 *for matrices of real elements,* $P=N$ $(P=N-1$ *for* CR-2).

times agree within 20 percent with the measurements, except for the PCR algorithm. The measured times for this algorithm are considerably higher than the predicted times. For the Intel iPSC/1 the predicted (and by far the measured) times for the PCR algorithm are always higher than those of any of the CR algorithms described here. With respect to performance, odd-even cyclic reduction is always preferable to parallel cyclic reduction on the Intel iPSC/1.

The complexity of algorithm SS/TGET$(n-2)$ is always lower than that of algorithm SS/TGET$(n-1)$, if $N \geqq 4$. If one exchange operation takes the same time as one *send* or *receive* operation, then algorithm SS/BCGE(i) is always of lower complexity than algorithm SS/TGET(i). With respect to arithmetic time, the break-even point between the algorithms TGET(i) and SS/TGET(i) (for $i < n$) is at $N = 17/4$, independent of P; substructuring should always be performed, except for $N = 2$ or 4. With respect to data-transfer time, algorithm SS/TGET$(n-2)$ is of lower complexity than algorithm TGET$(n-2)$, if $P > N \geqq 16$. If $N = 8$ the break-even point is $P = 24$. For $N = 4$ algorithm TGET$(n-2)$ is always of lower complexity than algorithm SS/TGET$(n-2)$. With respect to start-up time, an algorithm with substructuring requires four more start-ups than the corresponding algorithm without substructuring, assuming unlimited buffer size. However, for limited buffer size, the start-up time asymptotically becomes proportional to the data-transfer time.

For the hybrid algorithms we conclude:

- With increasing maximum buffer size the boundary between the regions for gathering and substructuring moves in a direction implying more steps of gathering instead of substructuring. For a very small buffer size cyclic reduction should be used.
- Increasing the significance of the arithmetic time implies a growing region for substructuring, and cyclic reduction.
- Increasing the data-transfer time implies a growing region for substructuring, and cyclic reduction.

For complex systems the number of real arithmetic operations increases faster than the data volume. Hence, the communication contributes a smaller fraction to the total execution time for a complex system than a real system. See [16] for measured and predicted execution times on the Intel iPSC/1.

3. Multiple tridiagonal systems. With complete freedom of distributing the systems of equations, the obvious choice is to allocate all equations of a single system to the same processor. Then, the systems can be solved locally by Gaussian elimination. No communication is required and the number of arithmetic operations is the minimum of the methods considered here. The solution is trivially parallel.

If the tridiagonal systems arise in connection with the solution of some form of partial differential equation in two or more dimensions, then other considerations may guide the allocation of lattice points to processors. With a one-dimensional partitioning of the lattice all tridiagonal systems may be allocated identically over the entire cube. This allocation of equations is assumed in this section. We will consider two-dimensional partitionings in the next section.

It follows from our analysis (and experiments) of single system solvers that parallel cyclic reduction and broadcasting followed by Gaussian elimination are not of interest for the multiple systems case. Repeated application of odd-even cyclic reduction for each system results in poor load balance, since the processor holding the middle equation is active in all reduction stages. *Balanced Cyclic Reduction* (BCR) [14] balances the load by performing the reduction such that for a set of N systems the reduction process converges to a unique processor for each system.

We consider three methods for solving multiple tridiagonal systems in this section: pipelined Gaussian elimination, combinations of transposition of systems of equations and substructuring, and balanced cyclic reduction. We also determine the optimum combination of these three algorithms as a function of machine parameters, the number of systems, and the size of each system.

3.1. Pipelined Gaussian elimination. In the case of two-way pipelined Gaussian elimination, a processor performs forward elimination on one system, sends the last equation of that system to the next processor, then continues with the first equation it has of the next system, etc. The two middle processors solve for the appropriate variables, and the backsubstitution proceeds in a fashion similar to the forward elimination. The number of communication start-ups can be reduced by communicating the data for several equations at once, causing a reduction in the arithmetic efficiency of the pipe of processors. (However, blocking of equations may increase the utilization of each processor, if the functional units of a processor are internally pipelined.)

Let $T_{2\text{GE-}p}(P, Q; N, t_a, t_c, \tau, B_m, i)$ be the time for pipelined two-way elimination with 2^i systems per packet. Then, the complexity of a pipelined two-way elimination with one system per packet is

$$\begin{aligned} &T_{2\text{GE-}p}(P, Q; N, t_a, t_c, \tau, B_m, 0)\\ &\quad=\left\{\frac{8P}{N}\left(Q+\frac{N}{2}-1\right)+3Q-5\right\}t_a+\left\{4\left(Q+\frac{N}{2}-2\right)+2\left\lceil\frac{12Q}{B_m}\right\rceil\right\}\tau\\ &\qquad+(56Q+16N-64)t_c, \end{aligned}\tag{10}$$

for $Q > 1$. During the forward phase, the number of floating-point operations is $5P/N$ per system per processor (the first and the last processors have five fewer operations). During the backward phase, the number of floating-point operations is $3P/N$ per system per processor. The two middle processors have three more operations per system due to the fact that six operations are required to solve the two-by-two system of the two middle processors. The first term in the expression for the number of start-ups contains a factor of four: a factor of two is due to forward and backward elimination, and another factor of two is due to the sending *and* receiving of data during the pipelining. Four start-ups are subtracted totally for the first and the last step of the pipelining, for both forward and backward phases. It is assumed that $B_m \geqq 12$. The second term in the expression for the number of start-ups accounts for the exchange of the $3Q$ floating-point numbers between the two middle processors. It is assumed that the exchange is done for all the Q systems together. The number of element transfers are

$$2\cdot 4(3+1)\left(Q+\frac{N}{2}-1-1\right)+2\cdot 4\cdot 3Q.$$

For 2^i systems per packet, $0 \leqq i < \log_2 Q$, the complexity estimate is

$$\begin{aligned} T_{2\text{GE-}p}(P, Q; N, t_a, t_c, \tau, B_m, i)&=\left\{\frac{8P}{N}\left(Q+2^i\left(\frac{N}{2}-1\right)\right)+3Q-5\cdot 2^i\right\}t_a\\ &\quad+\left\{2\left(\left\lceil\frac{12\cdot 2^i}{B_m}\right\rceil+\left\lceil\frac{4\cdot 2^i}{B_m}\right\rceil\right)\left(\frac{Q}{2^i}+\frac{N}{2}-2\right)\right.\\ &\quad\left.+2\left\lceil\frac{12Q}{B_m}\right\rceil\right\}\tau+(56Q+2^{i+4}N-2^{i+6})t_c. \end{aligned}\tag{11}$$

The number of element transfers are

$$2 \cdot 4(3+1) \cdot 2^i\left(\frac{Q}{2^i}+\frac{N}{2}-1-1\right)+2 \cdot 4 \cdot 3Q.$$

For $2^i = Q$, the complexity is

$$\begin{aligned} T_{2\text{GE-}p}(P, Q; N, t_a, t_c, \tau, B_m, \log Q) &= Q(4P-2)t_a+8Q(N+1)t_c \\ &\quad +\left\{\left(\frac{N}{2}+1\right)\left\lceil\frac{12Q}{B_m}\right\rceil+\left(\frac{N}{2}-1\right)\left\lceil\frac{4Q}{B_m}\right\rceil\right\}\tau. \end{aligned} \tag{12}$$

Note that the coefficients of τ and t_c in equation (11) do not specialize to the coefficients of τ and t_c in equation (12), because the communication changes from "send *and* receive" to "send *or* receive." Equation (11) specializes to equation (10) when $i=0$. When $N=2$ and $2^i = Q$, both (11) and (12) degenerate to two-way elimination on two processors without pipelining. The complexity is

$$Q(4P-2)t_a+2\left\lceil\frac{12Q}{B_m}\right\rceil\tau+24Qt_c.$$

The optimal value of 2^i is $\approx\sqrt{Q\tau/(Pt_a+4Nt_c)}$ assuming $B_m \geqq 12Q$, and

$$T_{2\text{GE-}p}(P, Q; N, t_a, t_c, \tau, B_m, i_{\text{opt}}) \approx \frac{8PQ}{N}t_a+2(N-3)\tau+56Qt_c+8\sqrt{Q\tau(Pt_a+4Nt_c)}.$$

3.2. Equation transposition and substructured elimination. If all the systems are identically distributed across the entire cube, then a transpose operation on the data structure results in the trivially parallel case. A transpose operation on the result of the solution to the equations may be required. This procedure is the multiple systems version of algorithm TGET (n) (or TGET ($n-1$)). The number of arithmetic operations is the same as in the pipelined case, but there is no inefficiency in the use of arithmetic units. With $Q \bmod N = 0$, all processors perform an equal amount of work in the solution process. Two-way elimination does not offer any reduction in arithmetic complexity, unlike in the single-system case.

The transposition of the set of equations gathers the equations of Q/N systems into a single processor (and different processors for different sets). Each step of the transposition doubles the number of equations of a system that are present in a processor (and reduces the number of different systems in a processor by a factor of two). It may be advantageous to perform substructuring after every few transpose steps to reduce the data volume that needs to be communicated.

The transposition can be performed through a sequence of exchanges in the different dimensions. Such an algorithm is optimal with communication restricted to one port at a time. With concurrent communication on all ports of every processor, the communication time can be reduced by a factor that is at most equal to the dimension of the (sub)cube in which the systems to be gathered are allocated [15]. Since each step for $Q \bmod N = 0$ is an exchange step, the transposition in the multiple-systems case requires at least twice the number of start-ups of the single-system case. Additional start-ups may be required for a limited buffer size B_m, since half of the data set being transposed takes part in every exchange.

The complexity of the initial substructuring is

$$T_{SS}(P, Q; N, t_a, t_c, \tau, B_m) = 17Q\left(\left\lceil \frac{P}{N} \right\rceil - 1\right) t_a + 2(16+4)Qt_c + 2\left(\left\lceil \frac{16Q}{B_m} \right\rceil + \left\lceil \frac{4Q}{B_m} \right\rceil\right)\tau, \tag{13}$$

and the complexity of one gather and one scatter step of the transpose algorithm is

$$\begin{aligned} &T_{\text{sbt}}(P, Q; N, t_c, \tau, B_m) \\ &= \frac{20QP}{N} t_c + \left(\left\lceil \left\lceil \frac{Q}{2} \right\rceil \frac{16P}{NB_m} \right\rceil + \left\lceil \left\lfloor \frac{Q}{2} \right\rfloor \frac{16P}{NB_m} \right\rceil + \left\lceil \left\lceil \frac{Q}{2} \right\rceil \frac{4P}{NB_m} \right\rceil + \left\lceil \left\lfloor \frac{Q}{2} \right\rfloor \frac{4P}{NB_m} \right\rceil\right)\tau. \end{aligned} \tag{14}$$

For i steps of the transpose operation starting with P equations followed by substructuring (assuming $Q \bmod 2^i = 0$), we have

$$\begin{aligned} &T_{\text{sbt}}(P, Q; N, t_c, \tau, B_m, i) + T_{SS}\left(P, Q; \frac{N}{2^i}, t_a, t_c, \tau, B_m\right) \\ &= 2i\left(\left\lceil \frac{8QP}{NB_m} \right\rceil + \left\lceil \frac{2QP}{NB_m} \right\rceil\right)\tau + \frac{20iQP}{N} t_c \\ &\quad + 17Q\frac{P}{N}\left(1 - \frac{1}{2^i}\right) t_a + 2\left(\left\lceil \frac{16Q}{2^i B_m} \right\rceil + \left\lceil \frac{4Q}{2^i B_m} \right\rceil\right)\tau + \frac{40QP}{2^i N} t_c, \end{aligned}$$

where the first row is the transpose time, and the second row the substructuring time. If the procedure is repeated recursively, then $P/N = 1$ for all applications (since the substructuring always reduces the number of remaining equations per processor to one per system), except possibly for the first. In the last application it may also be the case that substructuring is replaced by (two-way) Gaussian elimination (if N is 1 or 2).

In the above complexity expressions the time for local data movement is ignored. For some computers, such as the Intel iPSC/1, this approximation is very poor [17]. The transpose operation implies a recursive partitioning of data sets into smaller and smaller blocks. With a high start-up time for communication it may be desirable to form messages of several blocks to fill a communication buffer. Accounting for the local data motion time, or many start-ups because of many messages being smaller than B_m, can significantly alter the complexity of the algorithm. With a copy time of t_{cp} for one byte, the copy time can be accounted for by using the following expression:

$$\begin{aligned} &T_{\text{sbt}}(P, Q; N, t_c, \tau, B_m, t_{\text{cp}}, i) \\ &= 2\sum_{j=1}^{i}\left(\min\left(2^{j-1}\left\lceil \frac{16PQ}{2^j NB_m} \right\rceil \tau, \left\lceil \frac{8PQ}{NB_m} \right\rceil + T_{\text{cp-}f}\right)\right. \\ &\quad \left. + \min\left(2^{j-1}\left\lceil \frac{4PQ}{2^j NB_m} \right\rceil \tau, \left\lceil \frac{8PQ}{NB_m} \right\rceil + T_{\text{cp-}b}\right)\right) + \frac{20iQP}{N} t_c, \end{aligned} \tag{15}$$

where $T_{\text{cp-}f} = (8PQ/N)t_{\text{cp}}$ and $T_{\text{cp-}b} = (2PQ/N)t_{\text{cp}}$ are the copy times for half of the local array for the forward and backward phases, respectively. Note that if $16PQ/2^iN \geqq B_m$, then no extra overhead is required due to local data motion during transposition.

For the Intel iPSC/1 the copy time for the communications buffer of size 1k bytes is approximately equivalent to 4τ. Including the copy time yields an expression of the form $(1+2+4+4+\cdots+4)\tau$ compared to $(1+1+1+1+\cdots+1)\tau$ if the copy time is ignored, assuming $8PQ/N \leqq B_m$. This difference is large enough to affect the choice of solution method for multiple tridiagonal systems.

An alternative to the optimized transpose algorithm used for the estimates above is to use the router. However, on the Intel iPSC/1 matrix transposition by the router software is inferior by approximately a factor of five for large (relative to the packet size) matrices, and by two orders of magnitude for small matrices (see Table 2).

TABLE 2
Matrix transposition on the Intel iPSC/1 *for one-dimensional partitioning.*

P, Q	Cube dim.	Algorithm: Optimum	Algorithm: Router
1024	4	1.8 sec	8.7 sec
1024	5	1.3 sec	6.9 sec
128	4	0.058 sec	4.4 sec
128	5	0.075 sec	9.0 sec

For the arithmetic and communication parameters of the Intel iPSC/1, the complexity estimates of an $n-1$ step transpose followed by two-way elimination and the same algorithm with substructuring preceding the transpose are compared in Fig. 7 for up to 5-cubes. Measured times are given in Fig. 7. We conclude that substructuring is not advantageous for the Intel iPSC/1. Note the importance of including the copy time for a good agreement between measured and predicted times.

In the substructured as well as the nonsubstructured estimates above we assume that all the data for an equation is moved in the transposition. But in some instances it is possible to store the system matrix pretransposed, and only transpose the right-hand side and the solution vector.

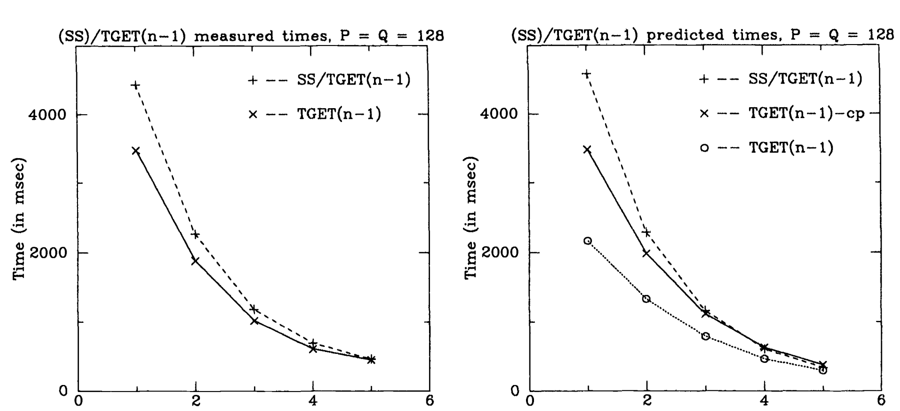


FIG. 7. *The measured and predicted times for the* TGET$(n-1)$ *and* SS/TGET$(n-1)$ *algorithms.* TGET$(n-1)$-cp *includes the copy time.*

3.3. Balanced cyclic reduction. The arithmetic complexity of the substructured elimination is the same as that of cyclic reduction, but cyclic reduction reduces the communication requirements. After the substructured elimination, each processor has one equation of each of the Q systems. In the BCR algorithm, the reduction process for each set of Q/N systems "converges" to a distinct processor, thereby keeping the load on the processors balanced. For each step of the reduction phase, the Q systems are partitioned into successively smaller subsets by virtue of the equations on which the reduction is performed. For instance, in the first step the set of equations is partitioned in half: in one half elimination is performed on even equations, in the other on odd equations. By numbering equations and systems from zero and letting the elimination on even equations be performed for even systems, the lowest-order bit in the binary encoding of the equation and system indices can be used to control the communication and elimination operations. Another obvious choice is to perform the elimination on even equations for the first half of the systems, in which case the highest-order bit in the system index and the lowest-order bit in the equation index are used for the control in the first step. By performing the operation recursively, the control proceeds toward higher/lower order bits.

The BCR algorithm can be implemented as an *exchange* algorithm with exchanges of equations between adjacent processors between elimination steps to keep all equations subject to further elimination in easily identifiable subcubes for each system. Such an exchange algorithm requires three exchanges for each step in the reduction phase, and three in the backsubstitution phase. Hence, with one send *or* one receive at a time, a total of approximately $12n$ start-ups is required, if the buffer size $B_m \geqq 8Q$ bytes for real systems. The scheme is then applied to both subcubes recursively. Note that four of these 12 communications can be saved, if the computations at each step are split into two parts—those depending on the preceding and succeeding row, respectively (as described for the CR-2 algorithm). Figures 8(a) and 8(b) show the four communication steps for one step of the BCR algorithm in the forward phase; Figs. 8(c) and 8(d) are another alternative of the same complexity.

The estimated times for the *exchange* version of the BCR and SS/BCR algorithms for $Q \bmod N = 0$ are

$$\begin{aligned} T_{\mathrm{BCR}}(N, Q; N, t_a, t_c, \tau, B_m) &= 17Q\left(1-\frac{1}{N}\right)t_a + \left\{4(16+4)Q\left(1-\frac{2}{N}\right)+2(16+4)\frac{Q}{N}\right\}t_c \\ &\quad + \left\{4\sum_{i=1}^{\log N-1}\left(\left\lceil\frac{16Q}{2^iB_m}\right\rceil+\left\lceil\frac{4Q}{2^iB_m}\right\rceil\right)\right. \\ &\quad \left. +2\left(\left\lceil\frac{16Q}{NB_m}\right\rceil+\left\lceil\frac{4Q}{NB_m}\right\rceil\right)\right\}\tau, \end{aligned} \tag{16}$$

$$\begin{aligned} &T_{\mathrm{SS/BCR}}(P, Q; N, t_a, t_c, \tau, B_m) \\ &\quad = T_{\mathrm{SS}}(P, Q; N, t_a, t_c, \tau, B_m) + T_{\mathrm{BCR}}(N, Q; N, t_a, t_c, \tau, B_m). \end{aligned} \tag{17}$$

For arbitrary Q and $P = N$, the complexity of one-step BCR is

$$\begin{aligned} T_{\mathrm{bcr}}(Q; t_a, t_c, \tau, B_m) &= 17\left\lceil\frac{Q}{2}\right\rceil t_a + 40Qt_c \\ &\quad + 2\left(\left\lceil\left\lceil\frac{Q}{2}\right\rceil\frac{16}{B_m}\right\rceil+\left\lceil\left\lfloor\frac{Q}{2}\right\rfloor\frac{16}{B_m}\right\rceil+\left\lceil\left\lceil\frac{Q}{2}\right\rceil\frac{4}{B_m}\right\rceil+\left\lceil\left\lfloor\frac{Q}{2}\right\rfloor\frac{4}{B_m}\right\rceil\right)\tau. \end{aligned} \tag{18}$$

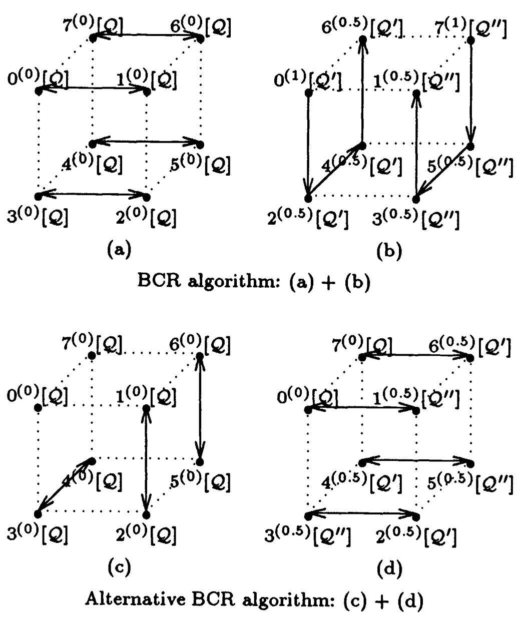


FIG. 8. *The four communication steps for one step of the* BCR *algorithm in the forward phase.* $\mathcal{Q} = \mathcal{Q}' \cup \mathcal{Q}''$ *is the set of* Q *systems. In* (a) *the left subcube sends its equations of the set* $\mathcal{Q}''$ *and receives its neighbor's equations of the set* $\mathcal{Q}'$. *In* (b) *equations* $1^{(0)}[\mathcal{Q}']$, $3^{(0)}[\mathcal{Q}']$, *and* $5^{(0)}[\mathcal{Q}']$ *are sent to equations* $2^{(0.5)}[\mathcal{Q}']$, $4^{(0.5)}[\mathcal{Q}']$, *and* $6^{(0.5)}[\mathcal{Q}']$, *respectively; the communication in the right subcube is similar.*

Hence, the complexity of i-steps of BCR for arbitrary Q can be derived by summing the complexity of each step with the first parameter being Q, $\lceil Q/2 \rceil$, $\lceil \lceil Q/2 \rceil /2 \rceil$, etc. An alternative to the exchange based BCR algorithm is to use an *in-place* algorithm. For such an algorithm we choose to use the router of the Intel iPSC/1. The result of implementing the two versions of BCR is shown in Fig. 9. The algorithm using the router requires 50–100 percent more time for solving the reduced systems on the Intel iPSC/1. The total time for the router based *in-place* algorithm is about 30 percent higher than that of the algorithm using exchanges for each step in the case of a 5-cube. For a fixed size problem the difference in the total time increases as the number of cube dimensions increases, since the substructuring part reduces in significance.

The predicted times for solving the reduced systems is lower than the measured times by approximately 30–70 percent for the 4- and 5-cubes, due mainly to the

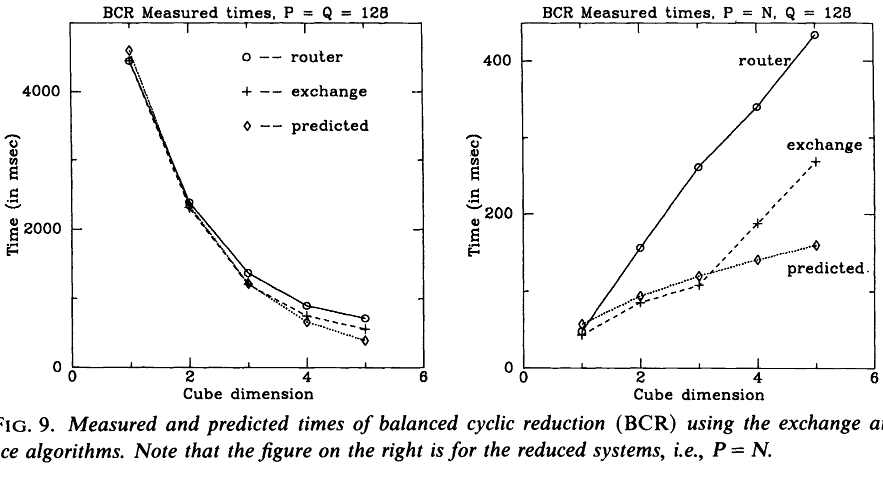


FIG. 9. *Measured and predicted times of balanced cyclic reduction* (BCR) *using the exchange and the in-place algorithms. Note that the figure on the right is for the reduced systems, i.e.,* $P = N$.

synchronization delay. In fact, an algorithm with each communication step involving all the processors tends to have a more significant delay for a larger cube. This characteristic is often the reason why the difference between the measured time and the predicted time increases with the cube size (Fig. 9).

With concurrent communication on all ports of every processor the number of start-ups can be reduced by a factor of two. Hence, each BCR step in the forward phase requires two communication steps (Fig. 8). This number of start-ups is minimal because in each step the processors that need to communicate are a distance two apart. However, it is possible to reduce the data-transfer time by a factor of up to four, (Fig. 8), by pipelining (a) and (b), or by overlapping the two alternatives ((a)+(b) and (c)+(d)) with each working on half of the systems. The overlapping algorithm does not cause any increase in the number of start-ups, and therefore is better than the pipelined algorithm.

3.4. Hybrid algorithms. The optimum choice of algorithm depends on machine parameters, the number of equations per system P, and the number of systems Q. In the balanced cyclic reduction method, and in the transpose and substructuring method, a problem is recursively changed into a number of problems on independent, smaller subcubes. The choice of algorithm should be reevaluated during the recursion process. The recursion can be stopped at any point by pipelined Gaussian elimination. The optimum method is determined by the following set of equations:

$$T_{\text{trid}}(P, Q; N, t_a, t_c, \tau, B_m)$$

$$\leqq \begin{cases} T_{2\text{GE}}(P, Q; 2, t_a, t_c, \tau, B_m) & \text{if } N=2, \\ \min\Big\{ T_{2\text{GE}}(N, Q; N, t_a, t_c, \tau, B_m), \\ \quad T_{\text{sbt}}(N, Q; N, t_c, \tau, B_m) + T_{\text{trid}}\left(N, \left\lceil \frac{Q}{2} \right\rceil; \frac{N}{2}, t_a, t_c, \tau, B_m\right), \\ \quad T_{\text{bcr}}(Q; t_a, t_c, \tau, B_m) + T_{\text{trid}}\left(\frac{N}{2}, \left\lceil \frac{Q}{2} \right\rceil; \frac{N}{2}, t_a, t_c, \tau, B_m\right)\Big\} & \text{if } P=N, \\ \min\Big\{ T_{2\text{GE}}(P, Q; N, t_a, t_c, \tau, B_m), \\ \quad T_{\text{sbt}}(P, Q; N, t_c, \tau, B_m) + T_{\text{trid}}\left(P, \left\lceil \frac{Q}{2} \right\rceil; \frac{N}{2}, t_a, t_c, \tau, B_m\right), \\ \quad T_{\text{SS}}(P, Q; N, t_a, t_c, \tau, B_m) + T_{\text{trid}}(N, Q; N, t_a, t_c, \tau, B_m)\Big\} & \text{otherwise,} \end{cases}$$

where

$$T_{2\text{GE}}(P, Q; N, t_a, t_c, \tau, B_m) = \min_{0 \leqq i \leqq \log_2 Q} \{T_{2\text{GE-}p}(P, Q; N, t_a, t_c, \tau, B_m, i)\},$$

$$T_{\text{SS}}(P, Q; N, t_a, t_c, \tau, B_m) = 17Q\left(\frac{P}{N}-1\right)t_a + 2\left(\left\lceil \frac{16Q}{B_m} \right\rceil + \left\lceil \frac{4Q}{B_m} \right\rceil\right)\tau + 40Qt_c,$$

$$T_{\text{sbt}}(P, Q; N, t_c, \tau, B_m) = 2\left(\left\lceil \frac{8PQ}{NB_m} \right\rceil + \left\lceil \frac{2PQ}{NB_m} \right\rceil\right)\tau + \frac{20QP}{N}t_c,$$

$$T_{\text{bcr}}(Q; t_a, t_c, \tau, B_m) = 17\left\lceil \frac{Q}{2} \right\rceil t_a + 4\left(\left\lceil \frac{8Q}{B_m} \right\rceil + \left\lceil \frac{2Q}{B_m} \right\rceil\right)\tau + 40Qt_c.$$

It is assumed that $Q \bmod 2 = 0$. If this is not true, then T_{sbt} becomes equation (14) and T_{bcr} becomes equation (18). $T_{2\mathrm{GE}\text{-}p}(P, Q; N, t_a, t_c, \tau, B_m, i)$ is the time for pipelined two-way elimination with 2^i systems per packet (eq. (11)). $T_{2\mathrm{GE}}$ is the time with optimal packet size 2^i.

Figure 10 shows the general calling sequencies for $Q = kN$. The label on an edge of the graph is defined as in Fig. 3. The two arguments of T_{SS}, and T_{sbt}, are P/N and Q. The arithmetic time of T_{SS}, and the data-transfer time of T_{sbt}, are proportional to

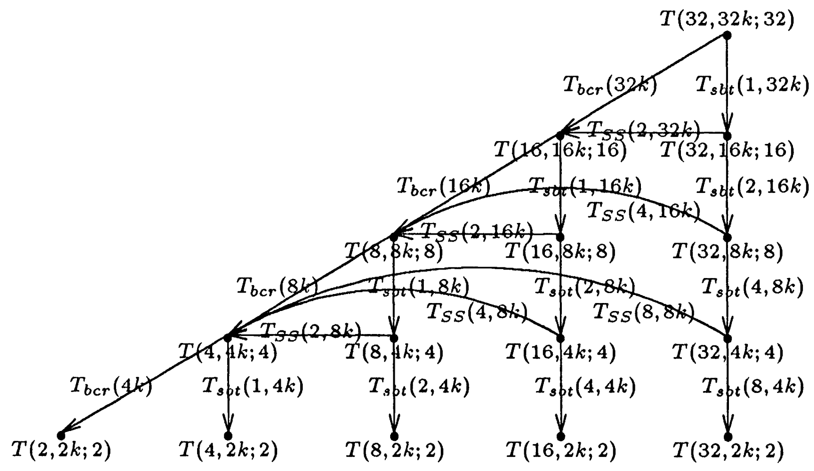


FIG. 10. *The general calling sequencies for* $32 \geqq P \geqq N$ *and* $Q = kN$.

the product of its two arguments. Figures 11(a)-(c) show how the calling sequencies depend on the maximum packet size for $t_a = 0$, $\tau = 1$, and $t_c = 0$. In the figure pipelined two-way Gaussian elimination with a block size of Q equations and one equation are represented by "∘" and "$*$" signs, respectively. When B_m is increased to ∞, part of the region for substructuring is replaced by transposition. Figures 11(d)-(f) give the optimum calling sequencies if both arithmetic time and data-transfer times are accounted for, and the communications overhead is high. Figure 12 shows the calling sequencies for a set of parameter values for which the value of B_m is irrelevant.

In the figures showing the optimum choice of algorithm assuming $t_{\mathrm{cp}} = 0$ the BCR algorithm is never used, whereas with a large value of t_{cp}, the BCR algorithm may be preferable, as seen from Fig. 11(e) $t_{\mathrm{cp}} = 0$ and Fig. 13 $t_{\mathrm{cp}} = 4$ (the value for the Intel iPSC/1). As B_m increases, the region for BCR extends to larger N along the $P = N$ diagonal line.

The time for i steps of the BCR algorithm is

$$\sum_{j=0}^{i-1} T_{\mathrm{bcr}}\left(\frac{Q}{2^j}; t_a, t_c, \tau, B_m\right) = 17Q\left(1 - \frac{1}{2^i}\right) t_a + 4 \sum_{j=0}^{i-1}\left(\left\lceil \frac{8Q}{2^j B_m} \right\rceil + \left\lceil \frac{2Q}{2^j B_m} \right\rceil\right)\tau + 80Q\left(1 - \frac{1}{2^i}\right) t_c, \tag{19}$$

assuming $P = N$ and $Q \bmod 2^i = 0$. If the time for local data motion can be ignored,

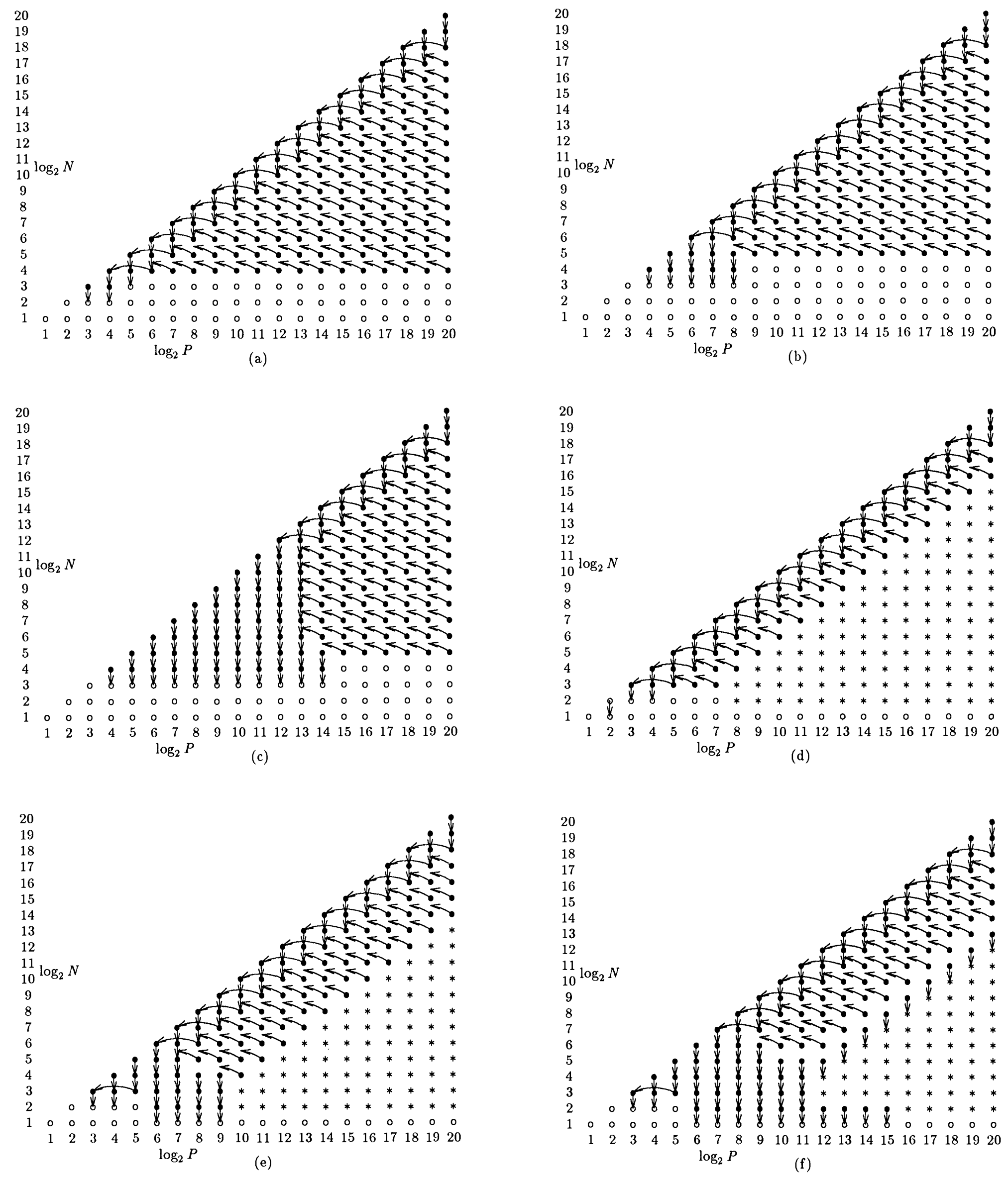


FIG. 11. *On the left, the calling sequencies for* $t_a = 0$, $\tau = 1$, $t_c = 0$, $t_{cp} = 0$, $Q = N$ *and* (a) $B_m = 16$, (b) $B_m = 1024$, *and* (c) $B_m = 2^{16}$. *On the right, the calling sequencies for* $t_a/t_c = 10$, $\tau/t_c = 1000$, $t_{cp} = 0$, $Q = N$ *and* (d) $B_m = 16$, (e) $B_m = 1024$, *and* (f) $B_m = 2^{16}$.

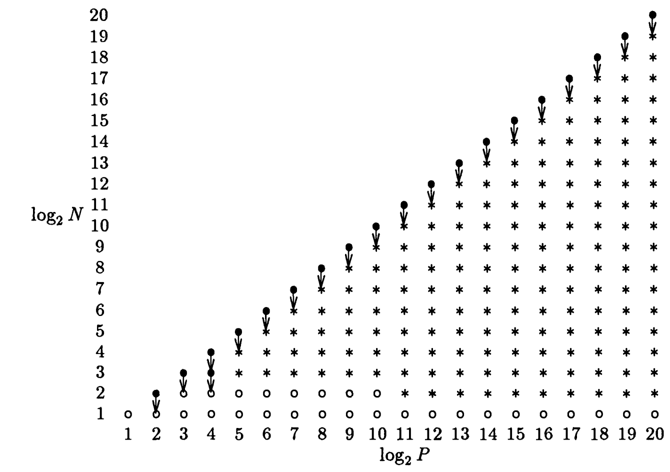


FIG. 12. *The calling sequencies for* $t_a/t_c = 0.01$, $\tau/t_c = 1$, $B_m \geqq 16$, $t_{\text{cp}} = 0$, *and* $Q = N$.

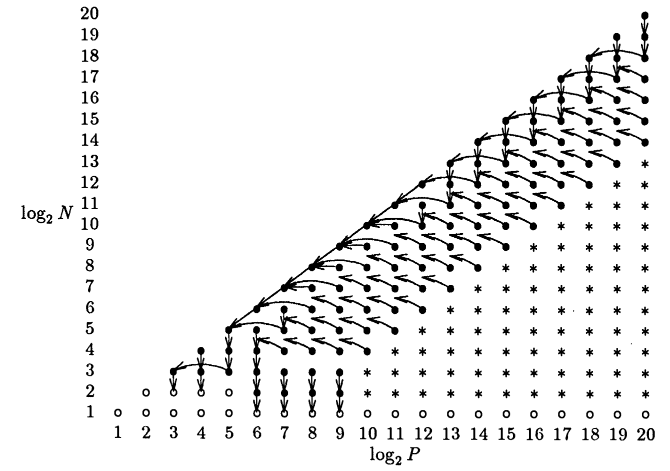


FIG. 13. *The calling sequencies for* $t_a/t_c = 10$, $\tau/t_c = 1000$, $B_m = 1024$, $t_{\text{cp}}/t_c = 4$, *and* $Q = N$.

then i-steps of the transpose algorithm followed by substructuring is

$$\sum_{j=0}^{i-1} T_{\text{sbt}}\left(N, \frac{Q}{2^j}; \frac{N}{2^j}, t_a, t_c, \tau, B_m\right) + T_{\text{SS}}\left(P, \frac{Q}{2^i}; \frac{N}{2^i}, t_a, t_c, \tau, B_m\right)$$

$$(20) \qquad = 17Q\left(1 - \frac{1}{2^i}\right) t_a + \left\{2i\left(\left\lceil \frac{8Q}{B_m} \right\rceil + \left\lceil \frac{2Q}{B_m} \right\rceil\right) + 2\left(\left\lceil \frac{16Q}{2^i B_m} \right\rceil + \left\lceil \frac{4Q}{2^i B_m} \right\rceil\right)\right\} \tau$$
$$+ \left(20iQ + \frac{40Q}{2^i}\right) t_c.$$

In order to account for the local data motion time the first term in the coefficients for both τ and t_c need to be changed according to equation (15).

Note that for $i = 1$, equations (19) and (20) yield the same complexity. For $i > 1$ equations (19) and (20) have the same arithmetic complexity. For $i > 1$ and $B_m \geqq 8Q$, the start-up time of (19) is always higher than that of (20). The ratio is two asymptotically. For a limited buffer size, the number of start-ups eventually compares as the data-transfer times. The data-transfer time of equation (19) is smaller than that of equation (20), except if $i = 2$ and 3. Hence, with $t_{cp} = 0$ the complexity of one step of BCR is the same as that of the one-step transposition followed by substructuring. The complexity of two steps (three steps) of BCR is always higher than that of two-step (three-step) transposition followed by substructuring, independent of the maximum buffer size. With $t_{cp} = 0$, BCR is never preferable over transposition and substructuring with respect to computational complexity for Q being a multiple of N. The optimum number of transposition steps for each substructuring depends on the values of t_a, τ, t_c, B_m, and Q. With local data motion requiring a significant time the BCR algorithm becomes competitive, especially for relatively large maximum packet size.

3.5. Summary of algorithms for solution of multiple systems. From the above analysis, and the recursion formulas for optimum choice of solution method we conclude that for multiple systems:

- Relatively high arithmetic time favors pipelined Gaussian elimination and transposition without substructuring.
- Relatively high data-transfer time favors pipelined Gaussian elimination instead of transposition.
- Relatively high communication start-up time favors transposition instead of pipelined Gaussian elimination.
- The block size for pipelined Gaussian elimination tends toward all equations when the arithmetic time is insignificant, otherwise the block size tends to be small (one equation) to maximize the use of the processor pipeline.
- A smaller communications buffer tends to favor substructuring instead of transposition.
- For a fixed N, doubling the number of equations almost doubles the arithmetic time for pipelined Gaussian elimination and leaves the communication time unchanged. With substructuring the total arithmetic time approximately doubles, but increases relatively somewhat more than for Gaussian elimination. The communication time is unchanged. Increasing P for fixed N and Q favors pipelined Gaussian elimination.
- Doubling both N and Q for fixed $P > 2N$ keeps the total work per processor constant. The start-up time and data-transfer times for pipelined Gaussian elimination approximately doubles. For transposition and substructuring the data-transfer time also approximately doubles, but there are only four additional start-ups. Hence, as N increases substructuring is favored.
- With insignificant time for local data motion the BCR algorithm is never competitive with the transpose/substructuring algorithm with respect to computational complexity. With a high cost for local data motion the conclusion may become the opposite because in the BCR algorithm $Q/2$ active systems out of Q can be selected to be continuous independent of the step. Increasing the maximum packet size favors the BCR algorithm more than the transpose/substructuring algorithm as far as copy time is concerned.
- The communication time of the transpose/substructuring algorithm can be sped up by at most a factor equal to the dimension of the subcube from which data are gathered (i for i-step transposition) with concurrent communication on all ports of every processor. A similar speed-up is not possible for the BCR algorithm, because of communication conflicts.

4. Two-dimensional decomposition. In the one-dimensional decomposition for the solution of multiple tridiagonal systems the choice of algorithm is either trivial (embarrassingly parallel), or the analysis above should guide the choice of algorithm. The nontrivial case is of particular interest for two- or higher-dimensional problems, and solution methods such as fast Poisson solvers and the Alternating Direction Method.

4.1. One directional solution. The one-dimensional analysis is easily generalized to the two-dimensional case by assuming that Q/N_2 systems are distributed to each of N_2 separate subcubes consisting of N_1 processors each, where $N_1 \times N_2 = N$. The allocation of partitions to processors within each subcube is made by a binary-reflected Gray code. The time for the solution of the systems of equations is given by $T_{\text{trid}}(P, Q/N_2; N_1, t_a, t_c, \tau, B_m)$, where $N = N_1 \times N_2$. It is fairly easy to verify that the solution time is minimized if N_2 is maximized. The one-dimensional partitioning yielding the "embarrassingly" parallel case is optimum.

4.2. Alternating Direction Methods (ADM). We consider a grid of $P \times Q$ internal points, and embed it in an $N_1 \times N_2$ processor mesh, that in turn is embedded in a Boolean n-cube by a two-dimensional binary-reflected Gray code. Thus, each processor is assigned PQ/N grid points. The two-dimensional Gray code ensures that each row and column of the processor mesh is itself a subcube, and that adjacency is preserved for each row and column. One ADM step consists of two half steps, each of which implies a number of tridiagonal matrix-vector multiplications and the solution of an equal number of tridiagonal systems. Each of the vectors in the matrix-vector multiplication represents the solution variables along a row (column) of the computational grid, and the tridiagonal matrix the approximation of derivatives along the same row (column). Similarly, tridiagonal systems are solved for each row (column). The second half step is the complement of the first—one forms the matrix vector products along the grid rows and solves tridiagonal systems along columns.

One ADM step consists of two half steps,

$$(I - \tfrac{1}{2}\Delta t A_x)u^{i+1/2} = (I + \tfrac{1}{2}\Delta t B_y)u^i,$$

$$(I - \tfrac{1}{2}\Delta t B_y)u^{i+1} = (I + \tfrac{1}{2}\Delta t A_x)u^{i+1/2}.$$

In each half step, computations are performed independently on grid rows or columns, i.e., on independent subcubes [19]. The matrix vector product takes a time of $5(PQ/N)t_a$ for the arithmetic, and requires exchanging Q/N_2 floating-point numbers with nearest (north and south) neighbors for one of the half step. This gives a total time of

$$(21) \qquad T_{mpy}\left(P, \frac{Q}{N_2}; N_1, t_a, t_c, \tau, B_m\right) = 5\frac{P}{N_1}\frac{Q}{N_2}t_a + 16\frac{Q}{N_2}t_c + \left\lceil \frac{4Q}{N_2 B_m} \right\rceil 4\tau,$$

for the matrix vector products of one half step. Then Q/N_2 tridiagonal systems of order P each are solved on subcubes of N_1 processors.

Let $T_{\text{ADM}}(P, Q; N, t_a, t_c, \tau, B_m)$ be the time for one step of ADM. Then,

$$(22) \qquad \begin{aligned} &T_{\text{ADM}}(P, Q; N, t_a, t_c, \tau, B_m) \\ &\leq \min_{0 \leq \log N_1 \leq \log N} \left\{ T_{\text{trid}}\left(P, \frac{Q}{N_2}; N_1, t_a, t_c, \tau, B_m\right) \right. \\ &\qquad + T_{\text{trid}}\left(Q, \frac{P}{N_1}; N_2, t_a, t_c, \tau, B_m\right) \\ &\qquad + T_{mpy}\left(P, \frac{Q}{N_2}; N_1, t_a, t_c, \tau, B_m\right) \\ &\qquad \left. + T_{mpy}\left(Q, \frac{P}{N_1}; N_2, t_a, t_c, \tau, B_m\right) \right\}. \end{aligned}$$

For matrix-vector multiplication and pipelined Gaussian elimination with one system per block, the leading term of the arithmetic complexity is independent of the shape of the processor array. A lower-order term in the arithmetic complexity for the pipelined Gaussian elimination is optimized for $N_1 = \sqrt{PN/Q}$. The number of start-ups for the matrix-vector multiplication as well as one of the two major terms in the expression for the number of start-ups in the pipelined Gaussian elimination are minimized at $N_1 = \sqrt{PN/Q}$. Another major term contributing to the number of start-ups in the pipelined Gaussian elimination algorithm is minimized for $N_1 = \sqrt{N}$. The optimum choice of N_1 for the data-transfer time is similar to that of start-up time. For a transpose algorithm followed by local Gaussian elimination, the complexity is independent of N_1 (for any B_m). For the SS/BCR algorithm, the arithmetic time is independent of N_1; the start-up time is independent of N_1 for large B_m, and minimized at $N_1 = \sqrt{PN/Q}$ for small B_m; the data-transfer time is minimized at $N_1 = \sqrt{PN/Q}$. With this value of N_1, $P/Q = N_1/N_2$, i.e., the aspect ratio of the processor grid is the same as that of the physical grid.

Let

$$\check{N}_1 = 2^{\lfloor (\log N + \log P - \log Q)/2 \rfloor} \quad \text{and} \quad \hat{N}_1 = 2^{\lceil (\log N + \log P - \log Q)/2 \rceil}$$

be the two (or one) power-of-two numbers closest to $N_1 = \sqrt{PN/Q}$. Let $\tilde{N}_1$ be the choice of $\check{N}_1$ and $\hat{N}_1$ that yields the smallest execution time, and let $N_{1_{\text{opt}}}$ be the optimal value of N_1 for equation (22) with T_{mpy} ignored (most of the time is spent on the tridiagonal part). Also let $\tilde{N}_2 = N/\tilde{N}_1$ and $N_{2_{\text{opt}}} = N/N_{1_{\text{opt}}}$. Using $\tilde{N}_1$ instead of $N_{1_{\text{opt}}}$ results in an execution time that is at most 20 percent higher than optimum for the set of machine parameters investigated earlier and $P = Q$. (The time for $\check{N}_1$ is the same as the time for $\hat{N}_1$ due to symmetricity.) The number of (P, N) pairs for which $\tilde{N}_1 \neq N_{1_{\text{opt}}}$ is a small fraction of the considered region of (P, N) pairs.

We also evaluated the nonoptimality of $\tilde{N}_1$ for $1 \leqq \log N \leqq \log P = \log Q - 4 \leqq 20$. In this case the largest deviation in execution time from the optimal value was 50 percent. Again, the region where the simplified choice results in measurable increase in execution time compared to the optimum choice is small.

The values of $\log N_{2_{\text{opt}}} - \log N_{1_{\text{opt}}} - \log \tilde{N}_2 + \log \tilde{N}_1$ for a few cases are shown in Fig. 14. The closer this number is to zero, the closer $N_{1_{\text{opt}}}$ is to $\tilde{N}_1$. The figure also shows the consequences of selecting an array aspect ratio as $\tilde{N}_1 \times \tilde{N}_2$ instead of $N_{1_{\text{opt}}} \times N_{2_{\text{opt}}}$ by giving the ratio of $r = T_{N_{1_{\text{opt}}}} / T_{\tilde{N}_1}$ where $T_{N_{1_{\text{opt}}}}$ and $T_{\tilde{N}_1}$ are the sum of the times for solving tridiagonal systems along the two axis with $N_1 = N_{1_{\text{opt}}}$ and $N_1 = \tilde{N}_1$, respectively. For Figs. 14(a) and 14(b), $r \geqq 0.86$ for the considered domain. If B_m in Figs. 14(a)-(d) is reduced to 16, then $N_{1_{\text{opt}}} = \tilde{N}_1$ for the considered domain.

Based on our examples, decreasing the maximum packet size, or increasing the data-transfer rate, tend to yield $N_{1_{\text{opt}}} = \tilde{N}_1$.

5. Conclusions. The execution time for multiprocessors with a packet switched communication systems and nodes without pipelined arithmetic units can be accurately modeled by the start-up time for a communication, the data-transfer time, maximum packet size, and the time for arithmetic operations. It may also be necessary to account for the local data motion time, as is the case for the Intel iPSC/1.

For a single tridiagonal system, the analysis as well as the experiments on the Intel iPSC/1 show that odd-even cyclic reduction can be competitive with parallel cyclic reduction with respect to arithmetic time, and in particular with respect to communication time. Odd-even cyclic reduction performed considerably better than parallel cyclic reduction on the Intel iPSC/1. A similar result has also been observed on the AMETEK system S14 [20].

$\log_2 N$

20																			0
19																		1	1
18																	0	0	0
17																1	1	1	1
16															10	0	0	0	0
15														9	9	1	1	1	1
14													8	8	8	0	0	0	0
13												1	7	7	7	1	1	1	1
12											0	6	6	6	6	0	0	0	0
11										1	1	5	5	5	5	1	1	1	1
10									0	0	4	4	4	4	4	0	0	0	0
9								1	1	1	3	3	3	3	3	1	1	1	1
8							0	0	0	0	2	2	2	2	2	0	0	0	0
7						1	1	1	1	1	1	1	1	1	1	1	1	1	1
6					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5				1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

$\log_2 P$

(a)

$\log_2 N$

20																			6
19																		5	1
18																	6	6	0
17																5	5	1	1
16															6	6	0	0	0
15														1	5	1	1	1	1
14													0	6	6	0	0	0	0
13												1	5	5	1	1	1	1	1
12											0	0	4	0	0	0	0	0	0
11										1	1	5	1	1	1	1	1	1	1
10									0	0	4	0	0	0	0	0	0	0	0
9								1	1	5	3	1	1	1	1	1	1	1	1
8							0	0	0	4	0	0	0	0	0	0	0	0	0
7						3	3	3	5	1	1	1	1	1	1	1	1	1	1
6					0	2	2	4	0	0	0	0	0	0	0	0	0	0	0
5				1	1	1	3	1	1	1	1	1	1	1	1	1	1	1	1
4			0	0	0	0	2	2	2	2	2	2	2	2	2	2	2	2	2
3		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

$\log_2 P$

(b)

$\log_2 N$

20																			0
19																		-1	-1
18																	0	0	0
17																1	-1	-1	-1
16															6	0	0	0	0
15														5	5	-1	-1	-1	1
14													4	4	4	0	0	0	0
13												3	3	3	3	-1	-1	1	-1
12											2	2	2	2	2	0	0	0	0
11										1	1	1	1	1	1	-1	1	-1	-1
10									0	0	0	0	0	0	0	0	0	0	0
9								1	1	1	-1	-1	-1	1	-1	-1	-1	-1	-1
8							0	0	0	0	0	0	0	0	0	0	0	0	0
7						1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1
6					-6	-6	-4	-4	-4	-4	-2	0	0	0	0	0	0	0	0
5				-5	-5	-5	-5	-5	-5	-3	-3	-3	-1	-1	-1	-1	-1	-1	-1
4			-4	-4	-4	-4	-4	-4	-4	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2
3		-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3
2	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

$\log_2 P$

(c)

$\log_2 N$

20																			-2
19																		-3	-3
18																	-2	-2	-2
17																-3	-3	-3	-3
16															-4	-4	-2	-2	-2
15														5	-3	-3	-3	-3	-3
14													4	-4	-2	-2	-2	-2	-2
13												3	-3	-3	-3	-3	-3	-3	-3
12											4	-4	-4	-4	-4	-4	-4	-4	-4
11										5	5	-3	-3	-3	-3	-3	-3	-3	-3
10									4	4	-2	-2	-2	-2	-2	-2	-2	-2	-2
9								3	3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3
8							2	2	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
7						1	1	1	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3
6					0	0	0	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2
5				-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
4			-4	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2
3		-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3
2	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

$\log_2 P$

(d)

$\log_2 N$

20																			1
19																		1	1
18																	1	1	1
17																1	1	1	1
16															.97	1	1	1	1
15														.93	.97	1	1	1	1
14													.93	.93	.96	1	1	1	1
13												.92	.92	.92	.96	1	1	1	1
12											.92	.92	.92	.92	.96	1	1	1	1
11										1	1	1	1	1	1	1	1	1	1
10									1	1	1	1	1	1	1	1	1	1	1
9								1	1	1	1	1	1	1	1	1	1	1	1
8							1	1	1	1	1	1	1	1	1	1	1	1	1
7						1	1	1	1	1	1	1	1	1	1	1	1	1	1
6					.89	.89	.73	.73	.73	.73	.91	1	1	1	1	1	1	1	1
5				.86	.86	.86	.67	.67	.67	.67	.83	.90	1	1	1	1	1	1	1
4			.80	.80	.80	.80	.80	.80	.80	1	1	1	1	1	1	1	1	1	1
3		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

$\log_2 P$

(e)

$\log_2 N$

20																			.99
19																		.99	1
18																	.96	.98	.99
17																.99	1	1	1
16															.96	.97	.98	.99	1
15														.98	.97	.99	.99	1	1
14													.96	.92	.96	.98	.99	.99	1
13												.97	.95	.97	.99	.99	1	1	1
12											.96	.92	.95	.97	.99	.99	1	1	1
11										.97	.98	.98	.99	.99	1	1	1	1	1
10									.92	.95	.93	.96	.98	.99	.99	1	1	1	1
9								.94	.96	.95	.98	.99	.99	1	1	1	1	1	1
8							.92	.93	.92	.96	.98	.99	.99	1	1	1	1	1	1
7						1	1	1	.98	.99	1	1	1	1	1	1	1	1	1
6					1	1	1	.97	.99	.99	1	1	1	1	1	1	1	1	1
5				1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4			.87	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

$\log_2 P$

(f)

FIG. 14. *The values of* $\log N_{2_{\rm opt}} - \log N_{1_{\rm opt}} - \log \tilde{N}_2 + \log \tilde{N}_1$ *for* (a) $t_a = 0$, $\tau = 1$, $t_c = 0$, $Q/P = 1$, (b) $t_a/t_c = 1$, $\tau/t_c = 1000$, $Q/P = 1$, (c) $t_a = 0$, $\tau = 1$, $t_c = 0$, $Q/P = 16$, *and* (d) $t_a/t_c = 1$, $\tau/t_c = 1000$, $Q/P = 16$; *all with* $B_m = 2^{16}$ *and* $t_{cp} = 0$. *The ratio of* $T_{N_{1_{\rm opt}}}/T_{\tilde{N}_1}$ *for the same sets of parameters as* (c) *and* (d) *are shown in* (e) *and* (f), *respectively.*

For multiple tridiagonal systems of equations the domain in the parameter space in which pipelined Gaussian elimination requires less time than the other considered algorithms is increasing at the expense of substructuring with increasing time for an arithmetic operation. The region for pipelined Gaussian elimination is increasing at the expense of transposition with or without substructuring with an increased time for the transfer of an element between processors. A high communication start-up time favors transpose based algorithms instead of pipelined Gaussian elimination. Reducing the size of the communications buffers favors substructuring and pipelined Gaussian elimination instead of transposition. The choice between substructuring and pipelined Gaussian elimination is generally a function of P/N, except for relatively small P and N for which pipelined Gaussian elimination is of choice. Increasing the size of individual systems for a fixed size cube and a fixed number of systems favors pipelined Gaussian elimination. Increasing the cube size and the number of systems for systems of fixed order favors substructuring. Balanced cyclic reduction is only of interest with a high cost for local data motion.

For multiprocessors with the performance characteristics of the Intel iPSC/1 and with at least 1024 processors (the Intel iPSC/1 is limited to at most 128 processors) pipelined Gaussian elimination would be the method of choice if $P/N \geqq 128$, transposition without substructuring the choice for $P/N \leqq 4$, and substructuring the preferred method otherwise, except for $P = N$ for multiprocessors with 32–4,096 processors. In the last case, balanced cyclic reduction is the preferred method with respect to execution time. For cubes with at most 512 processors, the choice of method is very case dependent (Fig. 13).

For Alternating Direction Methods choosing the aspect ratio of the processing array as close as possible to the aspect ratio of the physical domain is optimal for a large range of parameter values, and close to optimal for many other values.

Combining in the routing system is important for the performance of several of the algorithms. The required combining is of the merge/split type.

During the revision of the paper, some of our iPSC/1 codes were ported to iPSC/2 by Eisenstat and compared with the timing of his code using a different data structure [4]. A preliminary result shows that the codes by Eisenstat seem to perform slightly better than the codes by us for iPSC/2. Also, the choice of algorithm for iPSC/2 is different from that of iPSC/1.

Acknowledgments. We thank Stanley Eisenstat for his helpful comments on implementation issues and Eileen Daily for her editorial efforts.

REFERENCES

[1] B. L. BUZBEE, G. H. GOLUB, AND C. W. NIELSON, *On direct methods for solving Poisson's equations*, SIAM J. Numer. Anal., 7 (1970), pp. 627–656.

[2] M. Y. CHAN, *Dilation-2 embeddings of grids into hypercubes*, Tech. Report UTDCS 1-88, Department of Computer Science, University of Texas, Dallas, TX, 1988.

[3] W. J. DALLY, *Wire-efficient* VLSI *multiprocessor communication networks*, Tech. Report, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, September 1986.

[4] S. C. EISENSTAT, personal communication, 1988..

[5] A. GEORGE, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.

[6] C.-T. HO AND S. L. JOHNSSON, *Distributed routing algorithms for broadcasting and personalized communication in hypercubes*, in 1986 Internat. Conference on Parallel Processing, IEEE Computer Society, New York, 1986, pp. 640–648.

[7] ———, *On the embedding of arbitrary meshes in Boolean cubes with expansion two dilation two*, in 1987 Internat. Conference on Parallel Processing, Pennsylvania State University, 1987, pp. 188–191.

[8] ———, *Spanning balanced trees in Boolean cubes*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 607–630.

[9] R. W. HOCKNEY AND C. R. JESSHOPE, *Parallel Computers*, Adam Hilger, Bristol, 1981.

[10] S. L. JOHNSSON, *Odd-even cyclic reduction on ensemble architectures and the solution tridiagonal systems of equations*, Tech. Report YALE/DCS/RR-339, Department of Computer Science, Yale University, New Haven, CT, October 1984.

[11] ———, *Data permutations and basic linear algebra computations on ensemble architecture*, Tech. Report YALEU/DCS/RR-367, Department of Computer Science, Yale University, New Haven, CT, February 1985.

[12] ———, *Communication efficient basic linear algebra computations on hypercube architectures*, J. Parallel Distributed Comput., 4 (1987), pp. 133–172.

[13] S. L. JOHNSSON, *Fast* pfe *solvers on fine and medium grain architectures*, in Advances in Computer Methods for Partial Differential Equations—VI, IMACS, 1987, pp. 405-410.

[14] ———, *Solving tridiagonal systems on ensemble architectures*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 354-392.

[15] S. L. JOHNSSON AND C.-T. HO, *Spanning graphs for optimum broadcasting and personalized communication in hypercubes*, IEEE Trans. Comput., 38 (1989), pp. 1249-1268.

[16] ———, *Multiple tridiagonal systems, the alternating direction method, and Boolean cube configured multiprocessors*, Tech. Report YALEU/DCS/RR-532, Department of Computer Science, Yale University, New Haven, CT, June 1987.

[17] ———, *Matrix transposition on Boolean n-cube configured ensemble architectures*, SIAM J. Matrix Anal. Appl., 9 (1988), pp. 419-454.

[18] S. L. JOHNSSON AND P. LI, *Solutionset for* AMA/CS 146, Tech. Report 5085:DF:83, California Institute of Technology, Pasadena, CA, May 1983.

[19] S. L. JOHNSSON, Y. SAAD, AND M. H. SCHULTZ, *Alternating direction methods on multiprocessors*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 686-700; Tech. Report YALE/DCS/RR-382, Department of Computer Science, Yale University, New Haven, CT, August 1985.

[20] D. S. LIM AND R. V. THANAKIJ, *A survey of alternating direction implicit* (adi) *method implementations on hypercubes*, in Hypercube Multiprocessors 1987, Michael T. Heath, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

[21] A. RANADE AND S. L. JOHNSSON, *The communication efficiency of meshes, Boolean cubes, and cube connected cycles for wafer scale integration*, in 1987 Internat. Conference on Parallel Processing, IEEE Computer Society, New York, 1987, pp. 479-482.

[22] E. M. REINGOLD, J. NIEVERGELT, AND N. DEO, *Combinatorial Algorithms*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

[23] C. L. SEITZ, *Concurrent* VLSI *architectures*, IEEE Trans. Comput., 33 (1984), pp. 1247-1265.

[24] ———, *Experiments with* VLSI *ensemble machines*, J. VLSI Comput. Syst., 1 (1986), pp. 311-334.

[25] I. E. SUTHERLAND AND C. A. MEAD, *Microelectronics and computer science*, Scientific American, September 1977, pp. 210-228.

[26] H. H. WANG, *A parallel method for tridiagonal equations*, ACM TOMS, 7 (1981), pp. 170-183.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 3, pp. 593–599, May 1990

TIMELY COMMUNICATIONS

Under the "timely communications" policy for the SIAM Journal on Scientific and Statistical Computing, papers that have significant timely content and do not exceed five pages automatically will be considered for a separate section of the journal with an accelerated reviewing process. It will be possible for the note to appear approximately six months after the date of acceptance.

A FAN-IN ALGORITHM FOR DISTRIBUTED SPARSE NUMERICAL FACTORIZATION[*]

CLEVE ASHCRAFT[†], STANLEY C. EISENSTAT[‡], AND JOSEPH W. H. LIU[§]

Abstract. This paper presents a column-oriented distributed algorithm for factoring a large sparse symmetric positive definite matrix on a local-memory parallel processor. Processors cooperate in computing each column of the Cholesky factor by calculating independent updates to the corresponding column of the original matrix. These updates are sent in a fan-in manner to the processor assigned to the column, which then completes the computation. Experimental results on an Intel iPSC/2 hypercube demonstrate that the method is effective and achieves good speedups.

Key words. sparse Cholesky factorization, distributed numerical factorization, fan-in algorithm

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

1. Introduction. With the advent of local-memory parallel processors, many researchers have considered the problem of solving large sparse linear systems on such architectures [3], [4], [5], [7], [9], [11], [12]. In this paper, we present a new column-oriented distributed algorithm for computing the Cholesky factor of a large sparse symmetric positive definite matrix.

We assume that each processor has been assigned a subset of the columns in the matrix and is responsible for computing the corresponding set of columns in the Cholesky factor. We do not address the problem of how to symmetrically reorder the matrix to increase the potential parallelism, nor do we address the problem of how to assign columns to processors to balance the workload and reduce communication. See [7] for one approach to these problems.

Processors cooperate in computing each column of the Cholesky factor by calculating independent updates to the corresponding column of the original matrix. These updates are sent in a fan-in manner to the processor assigned to the column, which then completes the computation. Thus we refer to this as a *fan-in* algorithm for distributed sparse numerical factorization.

[*] Received by the editors June 5, 1989; accepted for publication October 17, 1989.

[†] Department of Computer Science, Yale University, New Haven, Connecticut 06520. The research of this author was supported in part by the Office of Naval Research under contract N00014-86-K-0310 and the National Science Foundation under grant DCR-85-21451.

[‡] 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 the Office of Naval Research under contract N00014-86-K-0310 and the National Science Foundation under grant DCR-85-21451.

[§] Department of Computer Science, York University, North York, Ontario, Canada M3J 1P3. The research of this author was supported in part by the Natural Sciences and Engineering Research Council of Canada under grant A5509.

$$\begin{pmatrix}
1 &&&&&&&&& \\
& 2 &&&&&&&& \\
&& 3 &&&&&&& \\
&&& 4 &&&&&& \\
& \bullet &&& 5 &&&&& \\
&& \bullet &&& 6 &&&& \\
\bullet & \bullet && \bullet & \bullet && 7 &&& \\
\bullet &&& \bullet &&& \bullet & 8 && \\
&&&&&&& \bullet & 9 & \\
\bullet && \bullet && \bullet & \bullet & \bullet & \bullet & \bullet & 10
\end{pmatrix}$$

FIG. 1. *A* 10×10 *symmetric matrix (only the lower triangle is shown).*

In §2, we briefly review Cholesky factorization by columns and introduce the notion of an aggregate update column. In §3, we describe how the fan-in algorithm uses aggregate update columns to compute the columns of the Cholesky factor. In §4, we present performance results on an Intel iPSC/2 hypercube. The speedups obtained compare favorably with those reported in [7] and [9].

2. Preliminaries. Let A be an $n \times n$ symmetric positive definite matrix and let L be its Cholesky factor, with entries a_{ij} and l_{ij}, respectively. The column-Cholesky method computes L column by column:

```
Algorithm 1: Column-Cholesky Factorization
    for column $j := 1$ to $n$ do
      begin
        $\begin{pmatrix} t_j \\ \vdots \\ t_n \end{pmatrix} := \begin{pmatrix} a_{jj} \\ \vdots \\ a_{nj} \end{pmatrix} - \sum_{k<j} l_{jk} \begin{pmatrix} l_{jk} \\ \vdots \\ l_{nk} \end{pmatrix}$
        $\begin{pmatrix} l_{jj} \\ \vdots \\ l_{nj} \end{pmatrix} := \frac{1}{\sqrt{t_j}} \begin{pmatrix} t_j \\ \vdots \\ t_n \end{pmatrix}$
      end
```

Here the temporary vector $(t_j, \cdots, t_n)^T$ is used only for clarity; its storage can overlap completely with that of $(l_{jj}, \cdots, l_{nj})^T$.

This formulation is applicable to both dense and sparse matrices. But in the sparse case, the updates $l_{jk}(l_{jk}, \cdots, l_{nk})^T$ to column j are sparse and come only from those preceding columns k of L for which $l_{jk} \neq 0$. These columns are given precisely by the nonzero structure of row L_{j*}:

$$Struct(L_{j*}) = \{k \mid k < j,\ l_{jk} \neq 0\}.$$

For example, consider the symmetric matrix whose lower triangle is shown in Fig. 1. Each nonzero in the matrix is represented by a "$\bullet$", and the matrix has been chosen so that no fill-in occurs during the factorization. The seventh column of the Cholesky factor is computed (sparsely) as

$$\begin{pmatrix} t_7 \\ t_8 \\ 0 \\ t_{10} \end{pmatrix} = \begin{pmatrix} a_{77} \\ a_{87} \\ 0 \\ a_{10,7} \end{pmatrix} - l_{71} \begin{pmatrix} \bullet \\ \bullet \\ \\ \bullet \end{pmatrix} - l_{72} \begin{pmatrix} \bullet \\ \\ \\ \\ \end{pmatrix} - l_{74} \begin{pmatrix} \bullet \\ \bullet \\ \\ \\ \end{pmatrix} - l_{75} \begin{pmatrix} \bullet \\ \\ \\ \bullet \end{pmatrix}.$$

Since $Struct(L_{7*}) = \{1,2,4,5\}$, both l_{73} and l_{76} are zero and columns 3 and 6 do not enter into the computation.

To describe the fan-in algorithm for distributed sparse numerical factorization, we now introduce four related notions:

- *factor column* $L_{*j} = (l_{jj}, \cdots, l_{nj})^T$;
- *update column* $l_{jk}(l_{jk}, \cdots, l_{nk})^T$, where $l_{jk} \neq 0$;
- *complete update column* $\sum_{k \in Struct(L_{j*})} l_{jk}(l_{jk}, \cdots, l_{nk})^T$;
- *aggregate update column* $\sum_{k \in K} l_{jk}(l_{jk}, \cdots, l_{nk})^T$, where $K \subseteq Struct(L_{j*})$.

The factor column L_{*j} is an implicit representation of the $n - j$ update columns

$$l_{ij}(l_{ij}, \cdots, l_{nj})^T, \quad i = j+1, \cdots, n.$$

Update columns and complete update columns are special cases of aggregate update columns where $K = \{k\}$ and $K = Struct(L_{j*})$, respectively.

In Algorithm 1, the column-Cholesky method is formulated in terms of complete update columns — the complete update column for column j is computed and subtracted from $(a_{jj}, \cdots, a_{nj})^T$.

It can also be formulated in terms of update columns — the update columns for column j are computed and subtracted one at a time. This is the approach used in a (nodal) sparse matrix factorization code, and in the distributed algorithm presented in [7] (where the update columns are sent implicitly as factor columns[1]).

And it can be formulated in terms of aggregate update columns — the update columns for column j are partitioned into disjoint sets, an aggregate update column is computed for each set, and these aggregate update columns are subtracted one at a time. This is the approach used in a supernodal sparse matrix factorization code [2] (where the groups correspond to supernodes), and is the basis for the fan-in algorithm presented in the next section.

3. A fan-in distributed algorithm.

Assume that we are given a mapping of columns to processors, and let $map[k]$ denote the processor assigned to column k. Then we can write the complete update column for column j as a sum of aggregate update columns (each corresponding to a different processor p):

$$\sum_{k \in Struct(L_{j*})} l_{jk}(l_{jk}, \cdots, l_{nk})^T = \sum_{p} u[j,p]$$

where

$$u[j,p] = \sum_{k \in row[j,p]} l_{jk}(l_{jk}, \cdots, l_{nk})^T,$$

and

$$row[j,p] = \{k \in Struct(L_{j*}) \mid map[k] = p\}.$$

Note that the update columns appearing in $u[j,p]$ all come from factor columns that are mapped to processor p. Thus $u[j,p]$ can be computed without any interprocessor communication. Of course, if $row[j,p] = \emptyset$, then $u[j,p] = 0$ so that

$$\sum_{k \in Struct(L_{j*})} l_{jk}(l_{jk}, \cdots, l_{nk})^T = \sum_{p \ni row[j,p] \neq \emptyset} u[j,p].$$

[1] That is to say, if $l_{jk} \neq 0$, then the factor column L_{*k} is sent to the processor assigned to column j, which then computes the update column $l_{jk}(l_{jk}, \cdots, l_{nk})^T$ and subtracts it from $(a_{jj}, \cdots, a_{nj})^T$.

We now state the fan-in algorithm for distributed numerical factorization.

Algorithm 2: Fan-in Distributed Cholesky Factorization

```
    $mycols = \{ j \mid map[j] = myname \}$ ;
    for column $j := 1$ to $n$ do
        if $row[j, myname] \neq \emptyset$ or $j \in mycols$ then
          begin
            $u := 0$ ;
            for $k \in row[j, myname]$ do
                $u := u + l_{jk}(l_{jk}, \cdots, l_{nk})^T$ ;
            if $j \notin mycols$ then
                Send aggregate update column $u$ to processor $map[j]$
            else
              begin
                $t := (a_{jj}, \cdots, a_{nj})^T - u$ ;
                while not all contributions have been received do
                    Receive an aggregate update column $u$ for column $j$
                      from another processor and subtract $u$ from $t$ ;
                $L_{*j} := t/\sqrt{t_j}$
              end
          end
```

Here *myname* denotes the processor-id of the processor under consideration, and all vector operations take advantage of sparsity.

This algorithm works for any mapping of columns to processors, although the performance is highly dependent on the precise mapping.

For simplicity, we have assumed that the aggregate column $u[j,p]$ is sent directly to the destination processor $map[j]$ and that t is formed by subtracting each of these contributions from $(a_{jj}, \cdots, a_{nj})^T$. While this approach works irrespective of the connectivity of the multiprocessor network, one could also take advantage of the underlying topology by partially summing the aggregate update columns on their way to the destination processor. With this change, the underlying logic closely resembles that used in the fan-in scheme of Romine and Ortega [10] for the solution of dense triangular linear systems.

To illustrate the operation of Algorithm 2, we use the example from §2. Assume that the columns have been assigned to processors p_1, p_2, and p_3 using a simple wrap mapping (see Fig. 2). The columns assigned to each processor are shown in Fig. 3.

$$\begin{array}{r c}
 & \left(\begin{array}{cccccccccc}
1 & & & & & & & & & \\
 & 2 & & & & & & & & \\
 & & 3 & & & & & & & \\
 & & & 4 & & & & & & \\
 & \bullet & & & 5 & & & & & \\
 & & \bullet & & & 6 & & & & \\
\bullet & \bullet & & \bullet & \bullet & & 7 & & & \\
\bullet & & & \bullet & & & \bullet & 8 & & \\
 & & & & & & & \bullet & 9 & \\
\bullet & & \bullet & & \bullet & \bullet & \bullet & \bullet & \bullet & 10
\end{array}\right) \\
\textbf{Processor} & \begin{array}{cccccccccc} p_1 & p_2 & p_3 & p_1 & p_2 & p_3 & p_1 & p_2 & p_3 & p_1 \end{array}
\end{array}$$

FIG. 2. *Wrap mapping of the columns of a* 10×10 *matrix.*

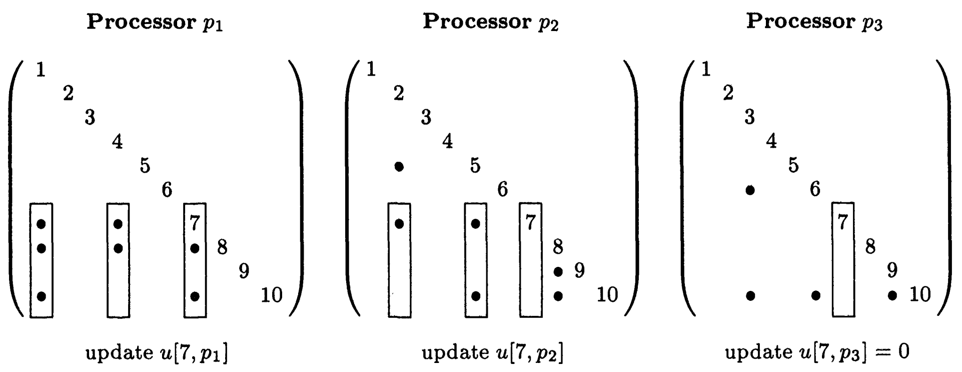


FIG. 3. *Fan-in factorization of column* 7.

Column 7 of the Cholesky factor belongs to processor p_1, and there are contributing update columns from processors p_1 and p_2, but not from processor p_3 (since both l_{73} and l_{76} are zero). Processor p_2 has to compute its contribution to column 7,

$$u[7,p_2] = l_{72}\begin{pmatrix} \bullet \\ \\ \\ \end{pmatrix} + l_{75}\begin{pmatrix} \bullet \\ \\ \\ \bullet \end{pmatrix},$$

and send it to processor p_1. Processor p_1 has to compute its contribution to column 7,

$$u[7,p_1] = l_{71}\begin{pmatrix} \bullet \\ \bullet \\ \\ \bullet \end{pmatrix} + l_{74}\begin{pmatrix} \bullet \\ \bullet \\ \\ \\ \end{pmatrix},$$

subtract the two aggregate update columns (sparsely),

$$\begin{pmatrix} t_7 \\ t_8 \\ 0 \\ t_{10} \end{pmatrix} = \begin{pmatrix} a_{77} \\ a_{87} \\ 0 \\ a_{10,7} \end{pmatrix} - u[7,p_1] - u[7,p_2],$$

and finally compute the factor column $(l_{7,7}, \cdots, l_{10,7})^T$.

4. Experimental results. The distributed fan-in algorithm for sparse Cholesky factorization was implemented in C and run on an Intel iPSC/2 hypercube with Weitek 1167 floating-point chips.

The test problems were nine-point finite-difference operators on rectangular grids. We used the nested dissection ordering [6], since it gives optimal-order fill and a well-balanced elimination tree. We used the subtree-to-subcube mapping [8] to assign processors to columns, since it gives good load balance and reduced communication. Table 1 contains the timing results and the corresponding speedups for three grid problems. The speedups are relative to a state-of-the-art serial code, again written in C.

Two other approaches to distributed numerical factorization are:

- the *fan-out* algorithm, in which each factor column is sent from its originating processor to every destination processor that needs it [7];

TABLE 1
Parallel factorization time and speedup on hypercube.

Problem	Serial	8 Processors		16 Processors		32 Processors	
	Time	Time	Speedup	Time	Speedup	Time	Speedup
31×31	2.74	0.54	5.07	0.39	7.03	0.32	8.56
63×63	23.26	3.86	6.03	2.41	9.65	1.56	14.91
125×63	60.21	9.26	6.50	5.67	10.62	3.52	17.11

- a distributed version of the *multifrontal* method [9].

The fan-out code achieved a speedup of 5.54 when solving an L-shaped grid problem with 2614 unknowns on a 16-processor hypercube [7]. The multifrontal code achieved a speedup of 9.5 when solving a nine-point problem for a 65×65 grid on a 16-processor hypercube [9]. The speedups given in Table 1 compare favorably with these results[2] and suggest the potential of the new distributed scheme.

However, one should not draw conclusions on the relative merits of these approaches based on these statistics, since the problems, machines, and baseline sequential codes differ. A thorough and detailed comparison of the fan-in, fan-out, and distributed multifrontal methods, and their respective implementations, will be given in [1].

Acknowledgment. We would like to thank Andy Sherman and the referees for their helpful comments and suggestions.

REFERENCES

[1] C. Ashcraft, S. Eisenstat, J. Liu, and A. Sherman, *A comparison of three distributed sparse factorization schemes*, presented at the SIAM Symposium on Sparse Matrices, Salishan Resort, Gleneden Beach, OR, May 1989.

[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] R. E. Benner, G. R. Montry, and G. G. Weigand, *Concurrent multifrontal methods: Shared memory, cache, and frontwidth issues*, Internat. J. Supercomputer Appl., 1 (1987), pp. 26–44.

[4] I. S. Duff, *Parallel implementation of multifrontal schemes*, Parallel Comput., 3 (1986), pp. 193–204.

[5] I. S. Duff, N. I. M. Gould, M. Lescrenier, and J. K. Reid, *The multifrontal method in a parallel environment*, Tech. Report CSS 211, Harwell Laboratory, Oxfordshire, England, 1987.

[6] A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.

[7] A. George, M. T. Heath, J. Liu, and E. Ng, *Sparse Cholesky factorization on a local-memory multiprocessor*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 327–340.

[8] A. George, J. Liu, and E. Ng, *Communication reduction in parallel sparse Cholesky factorization on a hypercube*, in Hypercube Multiprocessors 1987, M. T. Heath, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987, pp. 576–586.

[9] R. F. Lucas, *Solving planar systems of equations on distributed-memory multiprocessor*, Ph.D. thesis, Department of Electrical Engineering, Stanford University, Stanford, CA, Dec. 1987.

[10] C. H. Romine and J. M. Ortega, *Parallel solution of triangular systems of equations*, Parallel Comput., 6 (1988), pp. 109–114.

[2] We obtained similar speedups (6.8 and 10.6) with C implementations of these algorithms when solving a nine-point problem for a 63×63 grid on a 16-processor hypercube [1].

[11] E. ZMIJEWSKI, *Limiting communication in parallel sparse Cholesky factorization*, Tech. Report TRCS89-18, Department of Computer Science, University of California, Santa Barbara, CA, June 1989.

[12] E. E. ZMIJEWSKI, *Sparse Cholesky factorization on a multiprocessor*, Ph.D. thesis, Department of Computer Science, Cornell University, Ithaca, NY, August 1987.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 3, pp. 600-601, May 1990

ON THE USE OF STRUCTURAL ZEROS IN ORTHOGONAL FACTORIZATION*

JESSE L. BARLOW†

Abstract. If the orthogonal factorization of a sparse matrix A is the result of column updates, the numerical upper triangular factor R may have numerical nonzeros where there are structural zeros. This problem arises in the solution of sparse inequality-constrained least squares problems by active set methods.

In order to fit R into a static data structure, these nonzeros have to be neglected, even though, theoretically, they may be quite large. In this communication, it is shown that neglecting these nonzeros does not have an adverse effect.

Key words. least squares, symbolic factorization, sparsity, numerical nonzeros

AMS(MOS) subject classification. 65G05

Structural zeroes in orthogonal factorization. Let A be an $m\times n$ sparse matrix which has been factored into the form

$$A=QR, \tag{1}$$

where Q is an $m\times m$ matrix and R is an $m\times n$ matrix. R may be upper trapezoidal or it may be an intermediate stage in an orthogonal factorization of A. The matrix A is assumed to be a submatrix of the $m\times p$ matrix B, where $p\geqq n$. It is desired to allocate storage for the factorization of A based upon the symbolic factorization of B^TB from, say, SPARSPAK-B [2].

If the factorization is the result of a sequence of column updates, then we may have a factorization of the form

$$\hat{A}=A+\delta A=(Q+\delta\hat{Q})(R+\delta\hat{R})=\hat{Q}\hat{R}, \tag{2}$$

where $\hat{Q}=Q+\delta\hat{Q}$, $\hat{R}=R+\delta\hat{R}$ are the computed versions of Q and R in (1), δA is the backward error, and $\delta\hat{Q}$ and $\delta\hat{R}$ are forward errors in computing $\hat{Q}$ and $\hat{R}$. The entries of $\hat{R}$ may be nonzero where R has structural zeros according to the symbolic factorization of B^TB. This problem was encountered by Björck [1] in the solution of sparse box constrained least squares problems. To save storage and to be able to use a static data structure as in SPARSPAK-B [2], it is often convenient to neglect these numerical nonzeros in $\hat{R}$. Stewart (private communication) pointed out to Åke Björck that from his analysis [3], these nonzeros could be quite large. We show here that neglecting these values will not have a deleterious effect, no matter what their size.

First, we recognize that by setting structural zeros to zero, instead of storing $\hat{R}$ in (2) we are, in fact, storing $\tilde{R}$, which is defined by

$$\tilde{R}=R+\delta\tilde{R},\qquad \delta\tilde{R}=(\delta\tilde{r}_{ij}),$$

where the entries of $\delta\tilde{R}$ satisfy

$$\delta\tilde{r}_{ij}=\begin{cases}\delta\hat{r}_{ij} & \text{if } r_{ij}\neq 0 \quad \text{(by structural considerations)}\\ 0 & \text{if } r_{ij}=0.\end{cases}$$

* Received by the editors September 8, 1989; accepted for publication November 30, 1989. This research was supported by National Science Foundation grant CCR-8700172, the Air Force Office of Scientific Research grant AFOSR-88-0161, and Office of Naval Research grant N0014-80-0517.

† Department of Computer Science, Pennsylvania State University, University Park, Pennsylvania 16802.

Clearly

$$\|\delta\tilde{R}\|_F \leqq \|\delta\hat{R}\|_F.$$

Thus the forward error in $\tilde{R}$ is no greater than that of $\hat{R}$.

One concern would be how the use of these structural zeros would affect subsequent calculations. That is, suppose that R is an intermediate stage in an orthogonal factorization. Suppose that we also perform the orthogonal factorization

$$R = Q_2 U,$$

where U may be upper trapezoidal or just a later stage in the factorization. Then, computationally, we have

$$R + \delta\tilde{R} + \delta\tilde{R}_1 = \tilde{R} + \delta\tilde{R}_1 = (Q_2 + \delta\tilde{Q}_2)(U + \delta\tilde{U}),$$

$$R + \delta\hat{R} + \delta\hat{R}_1 = \hat{R} + \delta\hat{R}_1 = (Q_2 + \delta\hat{Q}_2)(U + \delta\hat{U}),$$

where $\delta\tilde{R}_1$ and $\delta\hat{R}_1$ are the backward errors in the second factorizations. From Stewart [3] the resulting forward errors in $\delta\tilde{Q}_2$ and $\delta\tilde{U}$ are

$$\|\delta\tilde{Q}_2\|_F \leqq \frac{3\kappa(R)\tilde{\omega}}{1 - 2\kappa(R)\tilde{\omega}} \tag{3}$$

and

$$\|\delta\tilde{U}\|_F \leqq \tilde{\omega} + \|\delta\tilde{Q}_2\|_F(1 + \tilde{\omega}). \tag{4}$$

Here $\tilde{\omega} = \|\delta\tilde{R} + \delta\tilde{R}_1\|_F / \|R\|_F$. We have analogous bounds for $\|\delta\hat{Q}_2\|_F$ and $\|\delta\hat{U}\|_F$. Since $\delta\hat{R}_1$ and $\delta\tilde{R}_1$ satisfy similar backward error bounds and the expressions (3) and (4) are monotone increasing functions in $\tilde{\omega}$, these bounds will not be adversely affected by setting numerical nonzeros to zero where there are supposed to be structural zeros. Indeed, they may be improved by doing so.

This result is supported by numerical tests in [1].

Acknowledgments. The author would like to thank Ake Björck for informing him of this problem. He would also like to thank the referees for their suggestions on the presentation.

REFERENCES

[1] A. Björck, *A direct method for sparse least squares problems with lower and upper bounds*, Numer. Math., 54 (1988), pp. 19-32.

[2] J. A. George and E. Ng, SPARSPAK: *Waterloo sparse matrix package user's guide for* SPARSPAK-B, Res. Report No. CS-84-37, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, November 1984.

[3] G. W. Stewart, *Perturbation bounds for the* QR *factorization of a matrix*, SIAM J. Numer. Anal., 14 (1977), pp. 509-518.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 4, pp. 603-620, July 1990

POTENTIAL FLOW IN CHANNELS*

L. GREENGARD†

Abstract. A method is presented for calculating potential flows in infinite channels. Given a collection of N sources in the channel and a zero normal flow boundary condition, the method requires an amount of work proportional to N to evaluate the induced velocity field at each source position. It is accurate to within machine precision and for its performance does not depend on the distribution of the sources. Like the Fast Multipole Method developed by Greengard and Rokhlin [*J. Comput. Phys.*, 73 (1987), pp. 325-348], it is based on a recursive subdivision of space, knowledge of the governing Green's function, and the use of asymptotic representations of the potential field. Previous schemes have been based either on conformal mapping, which experiences numerical difficulties with the domain boundary, or direct evaluation of Green's function. Both require $O(N^2)$ work.

Key words. fluid dynamics, potential flow, vortex method, N-body problem, Fast Multipole Method

AMS(MOS) subject classifications. 30B50, 31A15, 41A30, 65E05, 70C05, 70F10

1. Introduction. The evaluation of potential fields in infinite channels arises as a numerical problem in several areas, most notably electrostatics and fluid dynamics. The governing equation is the Poisson equation,

$$\Delta\Psi = -\xi \tag{1}$$

subject to an appropriate boundary condition. In this paper, we will restrict our attention to two-dimensional models and will consistently use the terminology of fluid dynamics. In viscous incompressible flow, the left-hand side is the stream function, the right-hand side is the vorticity, and the condition imposed on the boundary is that of zero normal flow

$$\mathbf{u}\cdot\mathbf{n} = 0, \tag{2}$$

where the velocity field $\mathbf{u}$ is given by

$$\mathbf{u} = \left(\frac{\partial\Psi}{\partial y}, -\frac{\partial\Psi}{\partial x}\right). \tag{3}$$

In terms of the stream function, this is equivalent to specifying homogeneous Dirichlet boundary conditions

$$\Psi = 0. \tag{4}$$

We will concentrate on particle models, where the vorticity field is discretized, not by a mesh, but by N point vortices,

$$\xi = \sum_{i=1}^{N} \xi_i \cdot \delta(x - x_i, y - y_i). \tag{5}$$

Here, δ is the Dirac δ-function and ξ_i is the strength of the ith point vortex located at (x_i, y_i). In vortex methods, what we would like to compute is the stream function and/or velocity field at each particle position. In the absence of boundary effects, the

* Received by the editors September 19, 1988; accepted for publication (in revised form) April 26, 1989. This work was supported in part by the Office of Naval Research under grant N00014-86-K-0310 and in part by a National Science Foundation Mathematical Sciences Postdoctoral Fellowship.

† Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, New York 10012.

desired results can be obtained from the free-space Green's function for the Poisson equation $(-(1/2\pi)\ln r)$ as follows:

$$\Psi(x_i, y_i) = \sum_{j\neq i} -\frac{\xi_j}{4\pi}\cdot \ln((x_i-x_j)^2+(y_i-y_j)^2) \quad \text{for } i=1,\cdots,N, \tag{6}$$

$$\mathbf{u}(x_i, y_i) = \sum_{j\neq i} -\frac{\xi_j}{2\pi}\cdot \frac{(y_i-y_j, x_j-x_i)}{(x_i-x_j)^2+(y_i-y_j)^2} \quad \text{for } i=1,\cdots,N. \tag{7}$$

Note that, using direct summation, the calculations (6) and (7) require an amount of work proportional to N^2. To overcome this obstacle, a variety of fast "N-body" methods have been proposed in the last few years, which reduce the computational complexity to $O(N \log N)$ or $O(N)$. These include particle-in-cell methods [1], [15], astrophysical tree codes [2], [3], series expansion methods [17], [20], and the fast multipole method [5], [9], [10], [18].

Remark 1.1. It is clear from (6) and (7) that the stream function and velocity field are unbounded in the neighborhood of a point vortex. In [7], Chorin introduced the idea of replacing the point vortices by "vortex blobs" whose induced field is held constant within a small neighborhood of the source. More recent work by Hald [11], Beale and Majda [4], and others has shown that higher-order accuracy can be obtained by using different approximations for the local field. Outside a finite-size core, however, the velocity field due to a vortex blob in most of these methods is simply that of a point vortex. Since we are interested in reducing the computational cost of vortex methods, which is generally dominated by *far field* interactions, we will ignore the precise nature of the local interactions and will continue to use the point vortex model.

For a straight channel, the fluid velocity cannot be obtained as in (6) and (7). The main difficulty is that the zero normal flow condition can only be satisfied by an infinite image system (§ 2), making direct summation over a collection of point sources impossible. The most commonly used technique for overcoming this problem in constrained flows is that of conformal mapping. By converting the calculation to one in the upper half plane, the boundary condition can be imposed with one image per particle, and the potential flow computed as in (6) and (7) with only double the number of point vortices (Fig. 1). An attractive feature of this approach is that the fast N-body algorithms for free-space calculations may be applied directly.

There are two objections to this mathematically reasonable procedure. In a channel with zero normal flow boundary conditions, the velocity field induced by a point source decays exponentially along the length of the channel. But the free-space Green's function used in the upper half plane decays as $1/r$, losing much of the physical

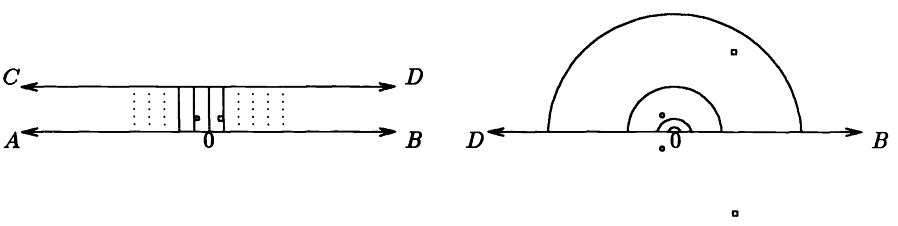


FIG. 1. *Conformal mapping of the channel to the upper half plane. The left-hand limit points A and C are mapped to the origin and the four solid vertical line segments in the channel are mapped to the four semicircles in the upper half plane. Two representative particles are marked by the small circle and square. The zero normal flow boundary condition is easy to apply with the method of images (each source is simply reflected across the x-axis and given opposite strength). Unfortunately, there is much stretching and contraction of the physical domain, which can cause practical difficulties.*

behavior of the solution. In fact, the physical behavior is expressed by the mapping itself, which for the strip $\{0 \leqq y \leqq \pi\}$ is simply e^z. The second objection is that a discretization of the boundary is often required (e.g., for vorticity generation). Conformal mapping, however, is well known to experience numerical difficulty when the derivative of the map has a great dynamic range [12], [16], [19]. This is clearly observed in Fig. 1, where the images of equispaced points along the top and bottom of the channel are points whose separation is growing (or contracting) exponentially. It would, on both counts, be much preferable to remain in the channel itself. To do this we will first need to replace the infinite image system by an analytic expression for the Green's function. This can be obtained through elliptic function theory. In [6], Choi and Humphrey derive expressions for both the infinite channel and a closed rectangular domain. With this expression, the velocity field can be obtained in a manner analogous to the N-body calculation of equation (7). Direct summation, of course, will require $O(N^2)$ work.

In this paper, we propose a new algorithm for two-dimensional potential flow in infinite channels. It is based on the analytically derived Green's function, and requires an amount of work proportional to N to evaluate all pairwise interactions. Like the Fast Multipole Method (FMM), it makes use of the superposition principle, far-field and local expansions, and a recursive subdivision of space. The channel algorithm consists of two distinct parts. The first part, described in §§ 2 and 3, is devoted to computing distant interactions along the length of the channel. After deriving the Green's function, we consider its long range behavior and define certain asymptotic representations of the far field that we refer to as stream expansions. We then carry out an initial decomposition of the computational domain along the length of the channel, and show how to efficiently compute interactions at a distance in the lengthwise direction. The second part of the algorithm is described in § 4, where we address the problem of computing near neighbor interactions. We show how elementary analysis can be used to reduce the computation to a set of uncoupled free-space problems, each of which can be solved by repeated application of the FMM.

2. Green's function for an infinite channel. We begin by developing an explicit expression for the velocity field induced by a point vortex in an infinite channel. The domain is defined to be the strip $\{0 \leqq y \leqq H\}$. We refer to the direction x increasing as *downstream* and to the direction x decreasing as *upstream*. We will use complex notation, equating the points (x_i, y_i) with the complex numbers z_i. If we define $\tilde{\mathbf{u}}$ by

$$\tilde{\mathbf{u}}(z_i) = \sum_{j \neq i} \frac{\xi_j}{2\pi} \cdot \frac{1}{z_i - z_j}, \tag{8}$$

then

$$\mathbf{u}(x_i, y_i) = (\operatorname{Im}(\tilde{\mathbf{u}}(z_i)), \operatorname{Re}(\tilde{\mathbf{u}}(z_i))) \tag{9}$$

is the velocity field induced by a collection of point sources with strength ξ_j located at the points $z_j = (x_j, y_j)$. In the remainder of this paper, we consider the calculation of $\tilde{\mathbf{u}}$ rather than $\mathbf{u}$ and will abuse notation by referring to $\tilde{\mathbf{u}}$ as the velocity field.

Let us now suppose that a source of unit strength is located inside the channel at z_0 and that z is a second point inside the channel with $z \neq z_0$. In order to satisfy the zero normal flow condition along the top and bottom of the channel, we introduce the infinite image system shown in Fig. 2.

Let us first add up the contributions from the images with positive strength, located at $z_0 + 2jHi$ $(j = -\infty, \cdots, \infty)$. The velocity field $\tilde{\mathbf{u}}_1(x)$ induced by these images is given

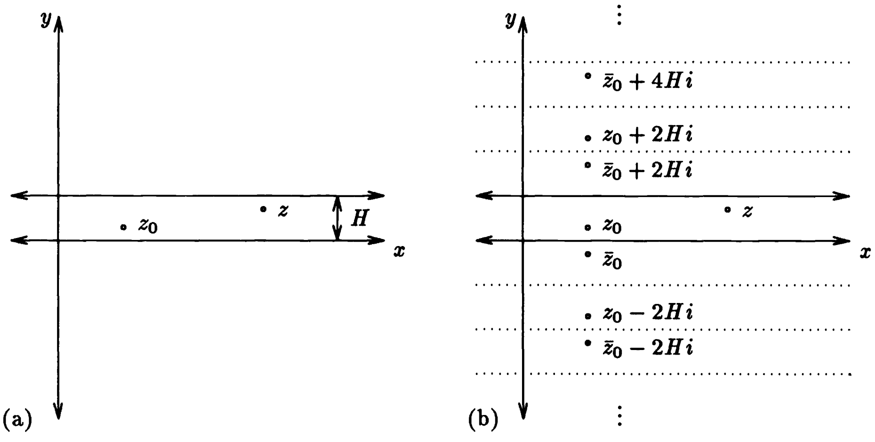


FIG. 2. *Enforcing boundary conditions by the method of images. Successive reflection across the top and bottom boundaries creates the image system shown. The images at positions* $\bar{z}_0+2jHi, j=-\infty,\cdots,\infty$ *have strengths of opposite sign from those at positions* z_0+2jHi.

by the expression

$$\tilde{\mathbf{u}}_1(z)=\sum_{j=-\infty}^{\infty}\frac{1}{z-z_0+2jHi} \tag{10}$$

$$=\frac{1}{z-z_0}+\sum_{j=1}^{\infty}\frac{1}{z-z_0+2jHi}+\frac{1}{z-z_0-2jHi} \tag{11}$$

$$=\frac{1}{z-z_0}+\sum_{j=1}^{\infty}\frac{2(z-z_0)}{(z-z_0)^2+4H^2j^2}. \tag{12}$$

But from ([8, p. 36]) we have

$$\coth(\pi z)=\frac{1}{\pi z}+\frac{2z}{\pi}\sum_{k=1}^{\infty}\frac{1}{z^2+k^2}, \tag{13}$$

so that

$$\tilde{\mathbf{u}}_1(z)=\sigma\cdot\coth(\sigma(z-z_0)), \tag{14}$$

where

$$\sigma=\frac{\pi}{2H}. \tag{15}$$

For the images with negative strength, located at $\bar{z}_0+2jHi$ ($j=-\infty,\cdots,\infty$), we obtain an induced velocity field $\tilde{\mathbf{u}}_2$, given by

$$\tilde{\mathbf{u}}_2(z)=-\sigma\cdot\coth(\sigma(z-\bar{z}_0)). \tag{16}$$

The net velocity field is, therefore,

$$\tilde{\mathbf{u}}(z)=\sigma\cdot(\coth(\sigma(z-z_0))-\coth(\sigma(z-\bar{z}_0))). \tag{17}$$

A simple integration yields the stream function Ψ induced by a point vortex,

$$\Psi(z)=\mathrm{Re}\left(\log\left(\frac{\sinh(\sigma(z-z_0))}{\sinh(\sigma(z-\bar{z}_0))}\right)\right). \tag{18}$$

A different derivation of Ψ is given by Choi and Humphrey in [6]. As mentioned previously, with this analytic expression for the pairwise interaction, the evaluation of the velocity field at each of the N source positions can be carried out in $O(N^2)$ operations. In order to develop a fast algorithm tailored to the channel itself, we need to examine the properties of the Green's function in more detail.

2.1. Upstream and downstream expansions. Let us suppose that z is downstream of z_0 ($\operatorname{Re}(z-z_0)>0$). Then

$$\coth(\sigma(z-z_0)) = \frac{e^{\sigma\cdot(z-z_0)}+e^{-\sigma\cdot(z-z_0)}}{e^{\sigma\cdot(z-z_0)}-e^{-\sigma\cdot(z-z_0)}} \tag{19}$$

$$= -1+\frac{2}{1-e^{-2\sigma\cdot(z-z_0)}} \tag{20}$$

$$= -1+2\cdot\sum_{k=0}^{\infty} e^{2\sigma z_0 k}\cdot e^{-2\sigma z k}. \tag{21}$$

Note that (21) can be obtained from (20) only if $e^{-2\sigma\cdot(z-z_0)}<1$, which is ensured by the condition that z be downstream of the source. From (17), then, the velocity field downstream of a unit source at z_0 has the expansion (about the origin)

$$\tilde{\mathbf{u}}(z) = 2\sigma\cdot\sum_{k=1}^{\infty}\left(e^{2\sigma z_0 k}-e^{2\sigma \bar{z}_0 k}\right)\cdot e^{-2\sigma z k}. \tag{22}$$

From this, it is immediately obvious that the decay in the field is exponential along the length of the channel. The main reason for developing an expansion of this form, however, is that it allows us to effectively use the superposition principle. By this we mean the following.

THEOREM 2.1. *Suppose that m sources with strengths* $\{q_j, j=1,\cdots,m\}$ *are located at points* $\{z_j, j=1,\cdots,m\}$, *with* $\operatorname{Re}(z_j)<r$. *Then for any point z further downstream* ($\operatorname{Re}(z)>r$), *the velocity* $\tilde{\mathbf{u}}(z)$ *induced by the sources is given by*

$$\tilde{\mathbf{u}}(z) = \sum_{k=1}^{\infty} a_k\cdot e^{-2\sigma z k} \tag{23}$$

where

$$a_k = 2\sigma\cdot\sum_{j=1}^{m} q_j\cdot\left(e^{2\sigma z_j k}-e^{2\sigma \bar{z}_j k}\right). \tag{24}$$

The error in truncating the expansion (23) *after p terms has the bound*

$$\left|\tilde{\mathbf{u}}(z)-\sum_{k=1}^{p} a_k\cdot e^{-2\sigma z k}\right| \leqq \frac{A\cdot x^{p+1}}{1-x} \tag{25}$$

where

$$A = 4\sigma\cdot\sum_{j=1}^{m}|q_j| \quad \textit{and} \quad x = e^{-2\sigma\cdot(\operatorname{Re}(z)-r)}. \tag{26}$$

Proof. The coefficients a_k are obtained directly from (22). The error bound is a consequence of the triangle inequality and the fact that (22) expands the field due to a single source as the sum of two geometric series. □

The upstream direction is treated in an analogous fashion. If $\operatorname{Re}(z-z_0)<0$, then the velocity due to a source at z_0 can be expressed as

$$\tilde{\mathbf{u}}(z)=2\sigma\cdot\sum_{k=1}^{\infty}\left(e^{-2\sigma z_0 k}-e^{-2\sigma \bar{z}_0 k}\right)\cdot e^{2\sigma z k}. \tag{27}$$

THEOREM 2.2. *Suppose that m sources with strengths* $\{q_j, j=1,\cdots,m\}$ *are located at points* $\{z_j, j=1,\cdots,m\}$, *with* $\operatorname{Re}(z_j)>r$. *Then for any point z further upstream* $(\operatorname{Re}(z)<r)$, *the velocity* $\tilde{\mathbf{u}}(z)$ *induced by the sources is given by*

$$\tilde{\mathbf{u}}(z)=\sum_{k=1}^{\infty} b_k\cdot e^{2\sigma z k} \tag{28}$$

where

$$b_k=2\sigma\cdot\sum_{j=1}^{m} q_j\cdot\left(e^{-2\sigma z_j k}-e^{-2\sigma \bar{z}_j k}\right). \tag{29}$$

The error in truncating the expansion (28) *after p terms has the bound*

$$\left|\tilde{\mathbf{u}}(z)-\sum_{k=1}^{p} b_k\cdot e^{2\sigma z k}\right|\leqq\frac{A\cdot x^{p+1}}{1-x} \tag{30}$$

where

$$A=4\sigma\cdot\sum_{j=1}^{m}|q_j| \quad \textit{and} \quad x=e^{2\sigma\cdot(\operatorname{Re}(z)-r)}. \tag{31}$$

DEFINITION 2.1. The expansions given by (28) and (23) will be referred to as *upstream* and *downstream* expansions, respectively. For a given collection of sources, the pair will be referred to as *stream expansions.*

The representation of the velocity field by means of these expansions may be viewed as an analogue of the multipole decomposition of the field due to a collection of sources in free space. It is important to keep in mind, however, that both their rate of decay and region of convergence are quite different.

Before examining the properties of stream expansions any further, we demonstrate their usefulness in computing far-field interactions with a simple example. For this, suppose that U_1 and U_2 are two sets, each containing N point vortices, located inside a channel of width H, and separated by a distance d (Fig. 3). To compute the velocity at each position in U_1 due to the sources in U_2 (or the velocity at each position in U_2 due to the sources in U_1) by means of the Green's function would require $O(N^2)$ operations. Let us instead form a downstream expansion due to the sources in U_1 and an upstream expansion due to the sources in U_2. From (25) and (30), it is easy to

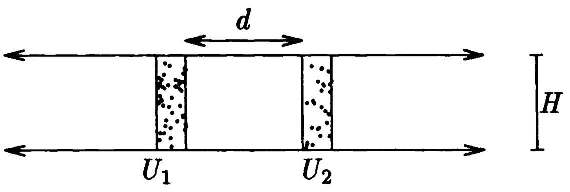


FIG 3. *Two clusters of point vortices located inside a channel of width H. The distance between the two clusters is denoted by d.*

determine a priori how many terms are needed to achieve a relative precision of ε. We simply require that

$$x^{p+1} \approx \varepsilon \quad \text{or} \quad p \approx \frac{H \cdot \ln(1/\varepsilon)}{\pi \cdot d}, \tag{32}$$

which is independent of N. The cost of formation of the two expansions is clearly proportional to Np. Evaluating the two expansions at all points in the relevant cluster also requires an amount of work proportional to Np, so that the total computation scales linearly with N, assuming that the relative precision ε and separation distance d are fixed.

2.2. The shifting lemma for stream expansions. The fast algorithm to be described depends not only on the formation and evaluation of stream expansions, but on their analytic manipulation. The following obvious lemma describes the necessary tools.

LEMMA 2.1. *Suppose that*

$$\tilde{\mathbf{u}}(z_u) = \sum_{k=1}^{\infty} b_k \cdot e^{2\sigma z_u k} \tag{33}$$

and

$$\tilde{\mathbf{u}}(z_d) = \sum_{k=1}^{\infty} a_k \cdot e^{-2\sigma z_d k} \tag{34}$$

are the upstream and downstream expansions about the origin due to m sources with strengths $\{q_j, j = 1, \cdots, m\}$, which are located at points $\{z_j, j = 1, \cdots, m\}$, with $-r <$ Re $(z_j) < r$ for some $r > 0$. Then

$$\tilde{\mathbf{u}}(z_u) = \sum_{k=1}^{\infty} \beta_k \cdot e^{2\sigma(z_u - z_0)k} \tag{35}$$

and

$$\tilde{\mathbf{u}}(z_d) = \sum_{k=1}^{\infty} \alpha_k \cdot e^{-2\sigma(z_d - z_0)k} \tag{36}$$

are the corresponding upstream and downstream expansions about z_0, where

$$\beta_k = b_k \cdot e^{2\sigma z_0 k} \tag{37}$$

and

$$\alpha_k = a_k \cdot e^{-2\sigma z_0 k}. \tag{38}$$

Furthermore, the error bounds for the shifted stream expansions are exactly the same as those for the original stream expansions.

Note that the behavior of shifted stream expansions contrasts sharply with that of multipole expansions in free-space (see [9], [10]). In the latter situation, the validity and accuracy of an expansion depends not only on the source positions but on the location of the expansion center. Note also from (24) and (29) that the coefficients of stream expansions about the origin are pure imaginary. If the centers of the shifted expansions are chosen to lie along the x-axis, then the coefficients in (37) and (38) are also pure imaginary, yielding a savings in both computational cost and storage.

Remark 3.1. To this point, we have been viewing stream expansions as representations of the *far field* due to a distribution of sources. It is possible, however, to view them in a different light. The expansions (33) and (34) of the preceding lemma are valid outside the strip $-r < \mathrm{Re}(z_j) < r$. By choosing a point z_0 upstream of the strip boundary ($\mathrm{Re}(z_0) < -r$), the shifted expansion (35) yields a representation of the induced field in a neighborhood of z_0. The same obviously holds for shifting a downstream expansion in the downstream direction (36). These are *local* representations of the field, the analogues of Taylor series in free-space, just as the far field stream expansions are the analogues of multipole expansions.

3. The Channel Decomposition Algorithm. In this section, we describe the first part of the fast algorithm. The basic idea is to subdivide the channel into vertical strips and to use stream expansions to compute far-field interactions.

The "elementary" strips into which the channel is refined have an aspect ratio of one third. The reason for not subdividing too much further is clear from equation (32). As d approaches zero, the number of terms required to achieve a fixed precision grows arbitrarily large. If we stop using expansions when $d = H/3$, however, then the number of terms required is given by

$$p \approx \frac{3 \cdot \ln(1/\varepsilon)}{\pi} < \ln\frac{1}{\varepsilon}. \tag{39}$$

We, somewhat arbitrarily, choose to stop subdividing the channel at this point. We will, of course, need to compute the interactions within an elementary strip and between nearest neighbor strips. This part of the calculation will be described in § 5. It relies on some additional analysis and the FMM for free-space problems.

The remainder of this section is devoted to a description of the channel decomposition algorithm. The main strategy used is that of clustering particles at a variety of spatial length scales and computing distant interactions by means of stream expansions. We begin by determining the locations of the extreme upstream and downstream particles. The corresponding section of the channel is considered to be the computational domain, and a sufficient number of elementary strips are created to cover the region (Fig. 4).

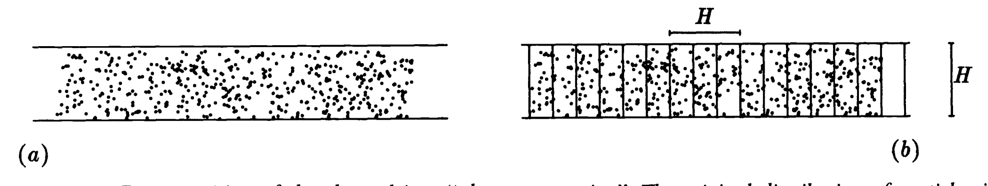


FIG 4. *Decomposition of the channel into "elementary strips." The original distribution of particles is shown in* (a). *In* (b), *a finite domain containing all particles has been subdivided into rectangular regions, each of which has an aspect ratio of one-third. S-expansions can be used to compute the interactions between particles contained in nonneighboring strips.*

We proceed by introducing a binary tree structure that groups particles at coarser and coarser levels (Fig. 5). Level 0 corresponds to the finest discretization of space (the elementary strips), whereas level $l+1$ is obtained from level l by the merger of two strips. The resulting strip at the higher level is referred to as the parent, while the two strips being merged are referred to as the children. Two strips at the same refinement level are said to be nearest neighbors if they share a boundary, otherwise they are said to be well separated. By construction, then, the minimum distance between well-separated strips is $H/3$, and to achieve a precision of ε in computing interactions via

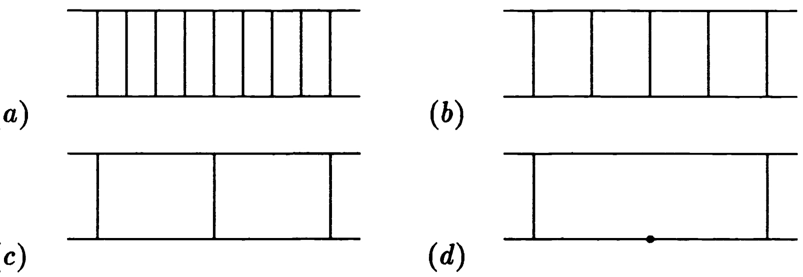


FIG 5. *In* (a), *eight elementary strips are shown that cover the computational domain. This level of spatial refinement is referred to as level* 0. *In* (b), (c), *and* (d), *pairs of strips are successively merged to form coarser and coarser subdivisions of the channel. The "center" of a strip is defined to be the midpoint of the segment of the x-axis bounded by that strip, as indicated in* (d).

stream expansions we need only choose the number of terms to be $p = \lceil \ln(1/\varepsilon) \rceil$. At coarser levels, the number of terms can obviously be decreased.

DEFINITION 3.1. The center of a strip is defined to be the midpoint of the segment of the x-axis bounded by that strip (Fig. 5(d)).

Other notation used in the description of the algorithm includes the following:

$F^u_{l,i}$ a p-term upstream expansion about the center of strip i at level l, describing the far field due to the particles contained inside the strip.

$F^d_{l,i}$ a p-term downstream expansion about the center of strip i at level l, describing the far field due to the particles contained inside the strip.

$L^u_{l,i}$ a p-term local stream expansion (see Remark 3.1) about the center of strip i at level l, describing the field due to all particles upstream of strip i's nearest neighbors.

$L^d_{l,i}$ a p-term local stream expansion (see Remark 3.1) about the center of strip i at level l, describing the field due to all particles downstream of strip i's nearest neighbors.

Interaction list for strip i at level l, it is the set of strips that are children of the nearest neighbors of i's parent and that are well separated from strip i (Fig. 6).

The channel decomposition algorithm is a two-pass procedure. In the first (upward) pass, we form the far-field stream expansions $F^u_{l,i}$ and $F^d_{l,i}$ for all strips at all levels, beginning at the level of elementary strips. In the second (downward) pass, we form the local stream expansions $L^u_{l,i}$ and $L^d_{l,i}$ for all strips at all levels, beginning at the coarsest level.

To see how the latter part is accomplished, suppose that at level $l+1$, the local expansions L^u and L^d have been obtained for each strip i. Then, by using Lemma 2.1 to shift these expansions to the centers of strip i's children, we obtain upstream and downstream expansions for each child strip at level l, describing the velocity field due to all particles upstream and downstream of strip i's nearest neighbors. For each strip j at level l, then, the interaction list is precisely that set of strips whose contribution

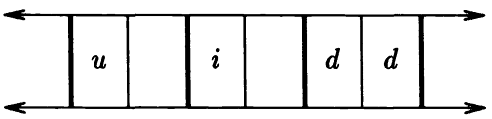


FIG 6. *The interaction list for strip i at level l. Strips marked with a "u" are upstream members of the list, whereas those marked with a "d" are downstream members of the list. Note that thick lines correspond to mesh level l+1.*

to the potential must be added to create $L^u_{l,j}$ and $L^d_{l,j}$ (Fig. 6). For each upstream member of the list, we use Lemma 2.1 to shift the center of the corresponding far-field expansion F^u to the center of strip j and add the result to the upstream expansion obtained from the parent. Similarly, for each downstream member of the list, we use Lemma 2.1 to shift the center of the corresponding far-field expansion F^d to the center of strip j and add the result to the downstream expansion obtained from the parent. Note that at the coarsest level, L^d and L^u are equal to zero, since there are no well-separated strips to consider.

Finally, for each strip j at the finest level, we evaluate the local expansions $L^d_{0,j}$ and $L^u_{0,j}$ at the position of each particle contained in the strip.

```
ALGORITHM 1.
Comment [Set number of terms to be used in expansions.]
    Choose the precision $\varepsilon$ to be achieved. Set the number of terms
    in all expansions to $p = \lceil \ln(1/\varepsilon) \rceil$.
                              Upward Pass
Step 1.
Comment [Decompose the channel into elementary strips.]
    Define elementary strip width to be $S_{wid} = H/3$.
    Compute $x_{\min}$ = $x$-coordinate of extreme upstream particle position.
    Compute $x_{\max}$ = $x$-coordinate of extreme downstream particle position.
    Compute number of elementary strips $K = \lceil (x_{\max} - x_{\min})/S_{wid} \rceil$.
    Compute height of binary tree $nlev = \lceil \log_2 K \rceil$.
Step 2.
Comment [Form far-field stream expansions at finest level.]
    do $i = 1, \cdots, K$
        Form $p$-term upstream and downstream expansions $F^u_{0,i}$ and $F^d_{0,i}$
        by using Theorems 2.1 and 2.2.
    end do
Step 3.
Comment [Form far-field stream expansions at all coarser refinement levels.]
    do $l = 1, \cdots, nlev$
        Form $p$-term upstream and downstream expansions $F^u_{l,i}$ and $F^d_{l,i}$
        for each strip $i$ at level $l$ by using Lemma 2.1
        to shift the center of each child strip's expansions to the current
        strip center and adding them together.
    end do
                              Downward Pass
Step 4.
Comment [Form local stream expansions at all refinement levels. Recall that $L^u$ and
$L^d$ are zero at level $nlev$ since there are no well-separated strips to consider.]
    do $l = nlev - 1, \cdots, 0$
        For each strip $i$ at level $l$, initialize $L^u_{l,i}$ and $L^d_{l,i}$
        by shifting the $L^u$ and $L^d$ expansions of strip $i$'s parent to the
        center of strip $i$. For each strip in $i$'s interaction list, determine
        whether it is upstream or downstream of strip $i$. If upstream, shift the
        center of the corresponding $F^u$ expansion to $i$'s center and add to
        $L^u_{l,i}$. If downstream, shift the center of the corresponding
        $F^d$ expansion to $i$'s center and add to $L^d_{l,i}$ (Fig. 6).
    end do
```

```
Step 5.
Comment [Local stream expansions are now available at the finest mesh level. They can be used to compute the velocity field due to all particles outside the nearest-neighbor elementary strips.]
    do $i = 1, \cdots, K$
        For each particle located in elementary strip $i$, evaluate
        $L^u_{0,i}$ and $L^d_{0,i}$. Add results together.
    end do
```

A brief operation count of the channel decomposition algorithm follows in Table 1. The estimate for the running time is therefore

$$N \cdot (4p+1) + K \cdot 6p. \tag{40}$$

4. The evaluation of near-neighbor interactions. The channel decomposition algorithm has left us with a sequence of uncoupled problems to consider. For each elementary strip, we must compute the internal interactions as well as the effects of the sources contained in that strip on the particles in the nearest neighbors (Fig. 7).

Because of their poor convergence rates in this regime, stream expansions are of limited use. We could proceed by direct evaluation of the remaining interactions through the use of the Green's function, but the asymptotic complexity of such an algorithm would be $O(N^2)$. Let us instead examine one of the subproblems in more detail.

We begin by reconsidering the method of images used to impose the zero normal flow condition in Fig. 2. Successive reflection across the top and bottom of the channel yields a one-dimensional array of squares (Fig. 8). These are either copies of the channel section itself or of its reflection across the bottom boundary, offset by $2jHi$ for some integer j. Note that we are only acting on the sources contained within the central elementary strip, but that we will compute the velocity field at particle positions within all three elementary strips of which the square is composed. In this manner, all interactions will have been accounted for exactly once.

TABLE 1

Step number	Operation count	Explanation
Step 1	order N	Examine each particle position to determine extreme upstream and downstream coordinates.
Step 2	order $2Np$	Each particle contributes to an upstream and a downstream expansion.
Step 3	order $K \cdot p$	The number of nodes in a binary tree is less than twice the number of leaves, so that the total number of nodes is of the order K. For each node, an amount of work of the order p is performed.
Step 4	order $5p \cdot K$	For each strip at each level, there are at most three entries in the interaction list. For each entry, the amount of work is proportional to p. In addition, two p-term expansions must be obtained from the parent.
Step 5	order $2Np$	Two p-term stream expansions are evaluated for each particle.

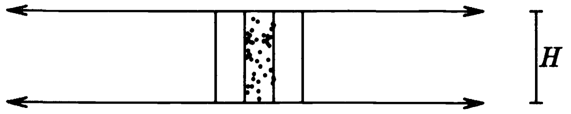


FIG 7. *In the second part of the algorithm, interactions are computed within each elementary strip and between nearest neighbors. This is accomplished by marching along the channel, considering one strip at a time, and accounting for its influence on all relevant particles.*

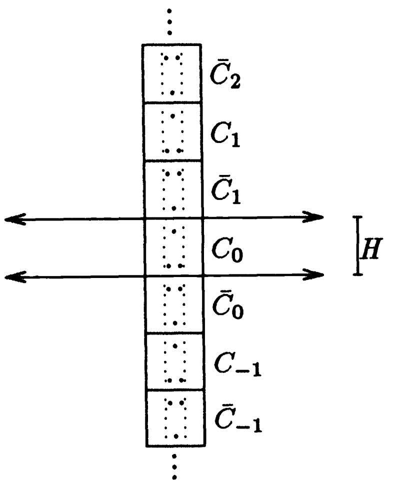


FIG. 8. *The channel section and its translated images are represented by boxes labeled C. The square obtained by reflection across the bottom boundary and its translates are labeled $\bar{C}$.*

The problem, again, is how to account for the sources in all images squares. We present a solution based on multipole expansions.

4.1. Multipole expansions. We will require two results. For the first, suppose that m point vortices with strengths q_i and positions z_i are located within a disk of radius r centered at the origin. Then, for a point z with $|z| > r$, the velocity field $\mathbf{v}(z)$ induced by the sources is given by a multipole expansion of the form

$$\mathbf{v}(z) = \sum_{k=1}^{\infty} \frac{a_k}{z^k}, \tag{41}$$

where

$$a_k = \sum_{i=1}^{m} q_i \cdot z_i^{k-1}. \tag{42}$$

The error in truncating the sum after s terms is

$$\left| \mathbf{v}(z) - \sum_{k=1}^{s} \frac{a_k}{z^k} \right| \leqq \left(\frac{A}{c-1} \right) \left(\frac{1}{c} \right)^s, \tag{43}$$

where

$$A = \sum_{i=1}^{m} |q_i| \quad \text{and} \quad c = \left| \frac{z}{r} \right|. \tag{44}$$

For a proof, see [9].

Note that to obtain a relative precision of ε (with respect to the total charge), the number of terms required in the series representation of $\mathbf{v}$ is approximately $-\log_c(\varepsilon)$, independent of m, the number of source charges.

The second result we need is contained in the following lemma, which describes the conversion of a multipole expansion into a local (Taylor) expansion inside a circular region of analyticity.

LEMMA 4.1 (Conversion of a Multipole Expansion into a Local Expansion). *Suppose that m sources of strengths $q_1, q_2, \cdots, q_m$ are located inside the circle D_1 with radius*

R and center at z_0, and that $|z_0| > (c+1)R$ with $c > 1$. Then the corresponding multipole expansion

$$(45)\qquad \mathbf{v}(z) = \sum_{k=1}^{\infty} \frac{a_k}{(z-z_0)^k},$$

converges inside the circle D_2 of radius R centered about the origin. Inside D_2, the potential due to the charges is described by a power series:

$$(46)\qquad \mathbf{v}(z) = \sum_{l=0}^{\infty} b_l \cdot z^l,$$

where

$$(47)\qquad b_l = \sum_{k=1}^{\infty} \frac{a_k}{a_0^{l+k}} \binom{l+k-1}{k-1} (-1)^k.$$

Furthermore, for any $s \geqq \max(2, 2c/(c-1))$, an error bound for the truncated series is given by

$$(48)\qquad \left| \mathbf{v}(z) - \sum_{l=0}^{s} b_l \cdot z^l \right| < \frac{A(4e(s+c)(c+1)+c^2)}{c(c-1)} \left(\frac{1}{c}\right)^{s+1},$$

where A is defined in (44) *and e is the base of natural logarithms.*

Proof. See [9] for the proof.

DEFINITION 4.1. Two squares with sides of length $2d$ are said to be *well separated* if they are separated by a distance $2d$.

Remark 5.1. Let A and B be well-separated squares with sides of length $2d$, and let D_A and D_B be the smallest disks containing the boxes A and B, respectively. Then the disks have radii $\sqrt{2} \cdot d$, and the distance from the center of one disk to the closest point in the other disk is at least $(4-\sqrt{2}) \cdot d$. Letting $c = (4-\sqrt{2})/\sqrt{2} \approx 1.828$, the error bound (48) applies with a truncation error using s-term expansions of the order c^{-s}.

Remark 5.2. In this section, the *center* of a square refers to its geometric center and not to its strip center (Definition 4.1.).

4.2. Reduction to a free-space problem. We will use Lemma 4.1 to account for all image sources outside the nearest-neighbor squares. The remaining calculation can then be carried using the free-space Green's function (see (6), (7)). We begin by choosing a coordinate system with the origin lying at the center of C_0. For each square C_j, the multipole expansion induced by the contained sources is of the form

$$(49)\qquad \mathbf{v}(z) = \sum_{k=1}^{s} \frac{a_k}{(z-z_j)^k},$$

where

$$(50)\qquad z_j = 2jHi$$

is the square's center. Note that the coefficients a_k of such a multipole expansion are translation invariant; i.e., they are identical for all integer j. Moreover, for $j \neq 0$, C_j is well separated from C_0, and the field induced inside the channel is accurately representable by an s-term local expansion, where $s = \lceil -\log_c(\varepsilon) \rceil$ is the number of terms needed to achieve a relative precision ε (see Remark 5.1.). This local representation is given by Lemma 4.1 as

$$(51)\qquad \Phi_j(z) = \sum_{m=0}^{p} b_m \cdot z^m$$

with

$$b_m = \sum_{k=1}^{p} \frac{a_k}{z_j^{m+k}} \binom{m+k-1}{k-1} (-1)^k. \tag{52}$$

Let S be the set of nonzero integers. To account for the field due to all well-separated images C_j, we compute the coefficients of a local representation by adding together the shifted expansions of the form (52) for all z_j with $j \in S$ to obtain

$$\Phi(z) = \sum_{m=0}^{p} b_m^{\text{total}} \cdot z^m \tag{53}$$

where

$$b_m^{\text{total}} = \sum_{k=1}^{p} a_k \binom{m+k-1}{k-1} (-1)^k \left(\sum_S \frac{1}{z_j^{m+k}} \right). \tag{54}$$

The summation over S for each inverse power of z_j can be precomputed and stored. For powers greater than one, the series is absolutely convergent. For $(m+k) = 1$, however, the series is not absolutely convergent, and the computed value depends on the order of addition. Choosing a reasonable value for the sum of the series requires consideration of the physical model. For this, suppose that the only particle in the simulation is a source of unit strength located at the origin. Then the image system corresponds to a uniform one-dimensional lattice, and by symmetry considerations, the induced velocity at any lattice point must be zero. But the net velocity of the particle at the origin corresponds to the summation over S of $1/z_j$, so that we set

$$\sum_S \frac{1}{z_j} = 0. \tag{55}$$

For powers $k > 1$, the summation over S of $1/z_j^k$ can be expressed in closed form by making use of the Riemann zeta function.

DEFINITION 4.2. The Riemann zeta function $\zeta(z)$ is defined by

$$\zeta(z) = \sum_{n=1}^{\infty} n^{-z}. \tag{56}$$

LEMMA 4.2. *For* $k > 1$,

$$\sum_S \frac{1}{z_j^k} = \begin{cases} 0 & \text{if } k \text{ is odd}, \\ \dfrac{(-1)^{k/2}}{2^{k-1} H^k} \zeta(k) & \text{if } k \text{ is even}. \end{cases} \tag{57}$$

Proof. The result follows immediately from the definitions of S and z_j. □

To account for the well-separated images of $\bar{C}_0$, we will require the corresponding multipole expansion. It is easy to verify, however, that for such squares, centered at a point w_j, the expansion is of the form

$$\mathbf{v}(z) = \sum_{k=1}^{s} \frac{\gamma_k}{(z - w_j)^k} \tag{58}$$

where

$$\gamma_k = -\bar{a}_k. \tag{59}$$

Except for $\bar{C}_0$ and $\bar{C}_1$, all of these images are separated from C_0, and as above, the fields they induce inside the channel section are accurately representable by a local expansion,

$$\Psi_j(z)=\sum_{m=0}^{p}\delta_m\cdot z^m \tag{60}$$

with

$$\delta_m=\sum_{k=1}^{p}\frac{\gamma_k}{w_j^{m+k}}\binom{m+k-1}{k-1}(-1)^k. \tag{61}$$

The well-separated images $\bar{C}_j$ clearly have centers

$$\cdots, -5Hi, -3Hi, 3Hi, 5Hi, \cdots. \tag{62}$$

Let T be the set of integers of the form

$$\{\pm(2j+1), j=1,2,\cdots,\infty\}. \tag{63}$$

We again account for the field due to all well-separated images by forming the coefficients of a local representation

$$\Psi(z)=\sum_{m=0}^{p}\delta_m^{\text{total}}\cdot z^m, \tag{64}$$

where

$$\delta_m^{\text{total}}=\sum_{k=1}^{p}\gamma_k\binom{m+k-1}{k-1}(-1)^k\left(\sum_T\frac{1}{w_j^{m+k}}\right). \tag{65}$$

The summation over T for $(m+k)>1$ is absolutely convergent. For $(m+k)=1$, the series is not absolutely convergent, but symmetry considerations again dictate the choice

$$\sum_T\frac{1}{w_j}=0. \tag{66}$$

For higher powers of k, a closed-form expression for the summation over T of w_j^k can be obtained through the use of Bernoulli numbers.

DEFINITION 4.3. The Bernoulli numbers B_n are given by the coefficients of $t^n/n!$ in the expansion

$$\frac{t}{e^t-1}=\sum_{n=0}^{\infty}B_n\cdot\frac{t^n}{n!}. \tag{67}$$

They satisfy the equation (see [8, p. 7])

$$\sum_{j=0}^{\infty}\frac{1}{(2j+1)^{2k}}=\frac{(2^{2k}-1)\pi^{2k}}{2\cdot(2k)!}|B_{2k}|. \tag{68}$$

LEMMA 4.3. *For $k>1$,*

$$\sum_T\frac{1}{w_j^k}=\begin{cases}0 & \textit{if } k \textit{ is odd},\\ \dfrac{(2^k-1)\pi^k|B_k|(-1)^{k/2}-2k!(-1)^{k/2}}{H^kk!} & \textit{if } k \textit{ is even},\end{cases} \tag{69}$$

where B_k is the kth Bernoulli number.

Proof. The result follows immediately from the definitions of T and w_j. □

If we add the computed coefficients δ_k^{total} from (65) to the coefficients b_k^{total} from (54), we obtain a single local expansion that describes the field due to *all* sources outside the nearest neighbor squares of C_0. This local expansion can then be evaluated at all particle positions in C_0.

The final step in the algorithm is to compute the velocity field due to the *free-spaces* sources within C_0, $\bar{C}_0$, and $\bar{C}_1$. This problem is handled by the FMM, which requires an amount of work proportional to $n+m$ to evaluate the field induced by n sources at m points.

ALGORITHM 2.

```
**Comment** [Set number of terms to be used in expansions.]
    Choose the precision $\varepsilon$ to be achieved. Set the number of terms
    in all expansions to $s = \lceil \log_c (1/\varepsilon) \rceil$.
**Comment** [From Algorithm 1, we are given that the number of elementary strips is $K$.]
    Define $n_i$ to be the number of particles in the $i$th strip.
    Clearly, $n_1 + n_2 + \cdots + n_K = N$, the total number of particles.
**Comment** [Process each elementary strip.]
    **do** $i = 1, \cdots, K$
        Define $C_0$ to be the square whose central third is strip $i$.
*Step* 1.
        Form coefficients $a_k$ of $s$-term multipole expansion about center
        of $C_0$ induced by sources in strip $i$.
        Form coefficients $\gamma_k$ of $s$-term multipole expansion for square $\bar{C}_0$
        via equation (59).
*Step* 2.
        Form coefficients $b_k + \delta_k$ of $s$-term local expansion about the
        center of $C_0$, which describes the field induced by all reflected
        sources outside the nearest-neighbor squares.
*Step* 3.
        Evaluate local expansion at all particle positions in strips $i-1, i$ and $i+1$.
*Step* 4.
        Compute velocity field induced by sources in $C_0$, $\bar{C}_0$, and $\bar{C}_1$
        at all particle positions in strips $i-1, i$ and $i+1$ via the FMM.
    **end do**
```

A brief operation count of Algorithm 2 follows in Table 2. The estimate for the running time is therefore

$$N \cdot (4s + 3\alpha) + K \cdot s^2. \tag{70}$$

TABLE 2

Step number	Operation count	Explanation
Step 1	order Ns	Each particle contributes to an s-term multipole expansion when its elementary strip is being processed.
Step 2	order Ks^2	The creation of a local expansion requires order s^2 work and is carried out once for each elementary strip.
Step 3	order $3Ns$	Three local expansions are evaluated for each particle.
Step 4	order $3\alpha N$	K free-space problems are solved, each of dimension n_i, with a factor of three included to account for the extra image sources and evaluation locations. The factor α represents the constant for the linear time FMM.

To summarize, then, the full algorithm consists of the following:

(1) Decomposition of the channel into elementary strips,

(2) **Algorithm 1** to compute distant interactions, leaving a sequence of uncoupled nearest-neighbor problems to consider,

Algorithm 2 to compute nearest-neighbor interactions.

5. Numerical results. A computer program has been implemented using the channel decomposition and nearest-neighbor algorithms of this paper. For testing purposes, we randomly assigned particles to positions within a channel section of length $5H$, where H was the channel width (Fig. 4), with source strengths between zero and one. Five-digit accuracy was requested from the expansions. In the first part of the algorithm, stream expansions were computed to 10 terms, while in the second part of the algorithm, multipole and Taylor expansions were computed to about 20 terms. We performed the calculations in four ways: (1) through the algorithm of this paper in single precision; (2) directly from the Green's function in single precision; (3) directly from the Green's function in double precision; (4) via conformal mapping in single precision. The direct evaluation from the Green's function in double precision was used as a standard for comparing the relative accuracies of the other three methods in a least squares sense. Calculations were carried out on a SUN 3/50 workstation using the 68881 coprocessor.

The following observations can be made from Table 3.

(1) The accuracies of the results obtained by the fast algorithm are in agreement with the error bounds given in this paper. In fact, the results are consistently more accurate than either of the direct calculations.

TABLE 3

Table of CPU *times in seconds required by the fast algorithm* (alg), *the direct Green's function method* (DIR), *and conformal mapping with direct evaluation of the resulting N-body problem* (cm). *The least squares errors for the three methods are shown in the last three columns. Timings in parentheses are estimated by computing the results for only a subset of* 100 *of the particles. The corresponding errors are computed from that smaller data set.*

N	T_{alg}	T_{dir}	T_{cm}	E_{alg}	E_{dir}	E_{cm}
100	8.38	34.8	14.0	$4.5 \cdot 10^{-7}$	$7.2 \cdot 10^{-7}$	$1.1 \cdot 10^{-6}$
400	53.1	551	223	$2.7 \cdot 10^{-7}$	$4.1 \cdot 10^{-7}$	$1.2 \cdot 10^{-6}$
1600	398	(8820)	(3550)	$4.3 \cdot 10^{-7}$	$1.3 \cdot 10^{-6}$	$1.1 \cdot 10^{-6}$
6400	1890	(141000)	(56800)	$6.9 \cdot 10^{-7}$	$5.2 \cdot 10^{-6}$	$3.4 \cdot 10^{-6}$

(2) The CPU time requirements of the fast algorithm appear to grow somewhat superlinearly. The reason for this is that there are two constants associated with the algorithm, a small one for the channel decomposition and a larger one for the FMM. The observed timings are dominated by the first constant for 100 and 400 particles, and by the second constant for the larger tests. When there are a small number of particles per strip, the FMM with its associated overhead is simply not invoked.

(3) By the time the number of particles reaches 6400, the fast algorithm is about 75 times more efficient than the direct Green's function method.

(4) Even for as few as 100 particles, the fast algorithm is about four times faster than the direct calculation.

6. Conclusions. A fast algorithm for potential flow in channels has been developed. It is based on asymptotic expansions that we refer to as stream expansions, some analytic observations concerning classical multipole expansions and Taylor series, and the fast multipole method. The asymptotic CPU time requirements for the algorithm grow linearly with the number of sources and, despite its complex structure, numerical experiments demonstrate that dramatic speedups can be obtained for evern moderate size particle systems.

In its current form, the algorithm requires that the channel boundaries be straight. A method applicable to channels with smoothly perturbed boundaries will be described in a subsequent paper. For polygonal (piecewise linear) perturbations of the channel, an attractive approach would be to conformally map the problem domain into an infinite strip. Howell and Trefethen [14] have recently developed a conformal mapping algorithm that can be used for just such purposes. A combination of their scheme with the method described in this paper should allow for large-scale simulations of practical interest in fluid dynamics and electrostatics.

A somewhat different generalization of obvious interest is that in which obstacles are present in the interior of the channel. For such calculations, the channel algorithm can be combined with the integral equation technique due to Rokhlin [18] to provide a fast method for computing potential flow around arbitrarily-shaped objects.

Acknowledgments. The author thanks V. Rokhlin for several useful discussions and the referees for suggestions that improved the presentation of the paper.

REFERENCES

[1] C. R. ANDERSON, *A method of local corrections for computing the velocity field due to a distribution of vortex blobs*, J. Comput. Phys., 62 (1986), pp. 111-123.

[2] A. W. APPEL, *An efficient program for many-body simulation*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 85-103.

[3] J. BARNES AND P. HUT, *A hierarchical $O(N \log N)$ force-calculation algorithm*, Nature, 324 (1986), pp. 446-449.

[4] J. T. BEALE AND A. MAJDA, *Vortex methods II: Higher order accuracy in two and three dimensions*, Math. Comp., 39 (1982), pp. 28-52.

[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] Y. CHOI AND J. A. C. HUMPHREY, *Analytical prediction of two-dimensional potential flow due to fixed vortices in a rectangular domain*, J. Comput. Phys., 56 (1984), pp. 15-27.

[7] A. J. CHORIN, *Numerical study of slightly viscous flow*, J. Fluid. Mech., 57 (1973), pp. 785-796.

[8] I. S. GRADSHTEYN AND I. M. RYZHIK, *Tables of Integrals, Series, and Products*, Academic Press, New York, 1980.

[9] L. GREENGARD AND V. ROKHLIN, *A fast algorithm for particle simulations*, J. Comput. Phys., 73 (1987), pp. 325-348.

[10] L. GREENGARD, *The Rapid Evaluation of Potential Fields in Particle Systems*, MIT Press, Cambridge, MA, 1988.

[11] O. HALD, *Convergence of vortex methods for Euler's equations*, SIAM J. Sci. Statist. Comput., 16 (1979), pp. 726-755.

[12] P. HENRICI, *Applied and Computational Complex Analysis, Vol.* 3, John Wiley, New York, 1988.

[13] R. W. HOCKNEY AND J. W. EASTWOOD, *Computer Simulation Using Particles*, McGraw-Hill, New York, 1981.

[14] L. H. HOWELL AND L. N. TREFETHEN, *A modified Schwarz-Christoffel transformation for elongated regions*, SIAM J. Sci. Statist. Comput., to appear.

[15] A. LEONARD, *Vortex methods for flow simulation*, J. Comput. Phys., 37 (1980), pp. 289-335.

[16] S. T. O'DONNELL AND V. ROKHLIN, *A fast algorithm for the numerical evaluation of conformal mappings*, Tech. Report 554, Computer Science Department, Yale University, New Haven, CT, 1986.

[17] A. M. ODLYZKO AND A. SCHÖNAGE, *Fast algorithms for multiple evaluations of the Riemann zeta function*, Trans. Amer. Math. Soc., 309 (1988), pp. 797-809.

[18] V. ROKHLIN, *Rapid solution of integral equations of classical potential theory*, J. Comput. Phys., 60, (1985), pp. 187-207.

[19] L. N. TREFETHEN, ED., *Numerical Conformal Mapping*, North-Holland, Amsterdam, 1986.

[20] L. VAN DOMMELEN AND E. A. RUNDENSTEINER, *Fast, adaptive summation of point forces in the two-dimensional Poisson equation*, J. Comput. Phys., 83, (1989), pp. 126-147.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 4, pp. 621-630, July 1990

A NEW SPARSITY PRESERVING QUASI-NEWTON UPDATE FOR SOLVING NONLINEAR EQUATIONS*

I. D. L. BOGLE† AND J. D. PERKINS‡

Abstract. A new quasi-Newton update is proposed based on a least relative change in the updated matrix, rather than on a least absolute change as is normally the case for quasi-Newton methods. The method corresponds to a quasi-Newton method weighted in a scale corresponding to a row scaling. The new method retains sparsity without having to include it as an affine transformation within the derivation. Unlike Broyden's and Schubert's methods, the method is scale-invariant. Comparisons are made on a set of problems that show promising results for the new method.

Key words. nonlinear equations, quasi-Newton, sparse, Jacobian, scale-invariant

AMS(MOS) subject classification. 65H10

1. Introduction.

In this paper we consider the problem

$$f(x)=0, \tag{1.1}$$

where f and x are vectors and f is a nonlinear function of x, with Jacobian J, which has a known sparsity pattern. Quasi-Newton methods have been developed for solving this problem when analytical Jacobian elements are not available. Broyden [4] first presented his method for full matrix problems, and a method in which a known Jacobian sparsity pattern is maintained was presented independently by Schubert [18] and Broyden [5] and analysed by Marwil [13]. In § 2 these two methods are reviewed. In § 3 the new sparsity preserving method is presented and derived. Finally, the three methods are compared on a set of test problems.

2. Quasi-Newton methods.

The theory of quasi-Newton methods, or least change secant methods as they are often called, is well known (Dennis and Schnabel [10]). An approximation A to the Jacobian $J(x)$ is maintained and updated. In both Broyden's and Schubert's method the updated matrix B satisfies the following condition:

$$\min \|B-A\|_F, \tag{2.1}$$

subject to certain constraints. For Broyden's method the only further condition required is the secant condition

$$B\in Q(y, s), \tag{2.2}$$

where

$$Q(y, s)=M(\in R^{n\times n}: Ms=y), \tag{2.3}$$

and

$$s=x_+-x \quad \text{and} \quad y=f_+-f$$

* Received by the editors August 10, 1987; accepted for publication (in revised form) June 7, 1989.

† Department of Chemical Engineering, University of Adelaide, P.O. Box 498, GPO Adelaide, South Australia 5001, Australia.

‡ Department of Chemical Engineering, Imperial College, Prince Consort Road, London SW7 2AZ, United Kingdom.

where if x and f correspond to the kth iteration then x_+ and f_+ correspond to the $(k+1)$th. Solving the constrained optimisation problem defined by (2.1) and (2.2) gives Broyden's update

$$B = A + \frac{(y - As)s^T}{s^T s}. \tag{2.4}$$

For sparse systems, where the sparsity pattern of the Jacobian is known, a third condition is added: that the sparsity pattern of A be maintained. Using the notation of Dennis and Schnabel, let $Z \in R^{n \times n}$ be the 0-1 matrix denoting the pattern of zeros in $J(x)$, i.e.,

$$Z_{ij} = \begin{cases} 0, & J(x)_{ij} = 0, \\ 1 & \text{otherwise,} \end{cases} \tag{2.5}$$

and let $SP(Z)$ denote the set of $n \times n$ matrices with this zero pattern, i.e.,

$$SP(Z) = (M \in R^{n \times n}: M_{ij} = 0 \text{ if } Z_{ij} = 0, 1 \leqq i, j \leqq n). \tag{2.6}$$

The update for sparse systems is defined by

$$\min \|B - A\|_F \quad \text{subject to } B \in Q \cap SP(Z). \tag{2.7}$$

Solving this constrained optimistion problem gives Schubert's update

$$B = A + P_z(D^+(y - As)s^T) \tag{2.8}$$

where $P_z: R^{n \times n} \to R^{n \times n}$ is the matrix projection operator defined by

$$(P_z(M))_{ij} = \begin{cases} 0, & Z_{ij} = 0, \\ M_{ij}, & Z_{ij} = 1. \end{cases} \tag{2.9}$$

Similarly for a vector $v \in R^n$, we define $\bar{v}_i \in R^n$ by

$$(\bar{v}_i) = \begin{cases} 0, & Z_{ij} = 0, \\ v_j, & Z_{ij} = 1, \end{cases} \tag{2.10}$$

where v_j is element j of the vector v, and the matrix D^+ is a diagonal matrix with elements

$$D^+_{ii} = \begin{cases} 1/d_i, & d_i \neq 0, \\ 0, & d_i = 0, \end{cases} \tag{2.11}$$

and where

$$d_i = \bar{s}_i^T \cdot \bar{s}_i.$$

Schubert's method is, therefore, an extension of Broyden method in which the change in the updated matrix is still minimized (in the Frobenius norm) but one further constraint is added: that of sparsity. The matrix projection operator P_z zeros out the elements of M corresonding to the zero positions of the sparsity pattern Z and leaves remaining elements of M unchanged. Schubert's method has not been found to be a reliable method for solving large systems of nonlinear equations. In spite of their similarity, Broyden's method has had much more success (Chen and Stadtherr [7], Perkins and Sargent [17]). A basic difference is that, although $\|B - A\|_F$ is minimised, because of the additional constraint of sparsity, the absolute change may be much greater in the sparse case.

Problem (2.7) can be extended to incorporate further information about the problem. Calamai and More [6] have recently shown that a sparse method for nonlinear equations may be derived that also satisfies constraints on the elements of the Jacobian approximation although no numerical results are given. An obvious inclusion is retaining symmetry of the updated matrix, useful for solving optimisation problems [9]. Interest in these methods has been limited because of the poor performance. Toint [19] has proposed a method that relies on the functions being partially separable, i.e., each equation is the sum of a small number of element functions, in which parts of the Jacobian corresponding to each element function are updated rather than updating the whole matrix. Toint reports some improvements for these methods over Schubert's method but the results are not conclusive.

Rather than minimize the absolute change in the matrices a new method is presented here that minimises a relative change in the matrix elements over the whole matrix, i.e.,

$$\min\left[\sum_{i=1}^{n}\sum_{j=1}^{n}\left(\frac{B_{ij}-A_{ij}}{A_{ij}}\right)^2\right]^{1/2}. \tag{2.12}$$

Equation (2.12) may be written in terms of the Frobenius norm as

$$\min\left\|\sum_{i=1}^{n}\sum_{j=1}^{n}\left(\frac{e_i^T(B-A)e_j}{e_i^TAe_j}\right)e_ie_j^T\right\|_F. \tag{2.13}$$

This produces a sparsity preserving update that has shown good numerical results on a variety of problems.

In the following section we derive the new update within the least change framework of Dennis and Schnabel [10]. Following this we compare its performance with the methods of Broyden and Schubert on a series of standard problems.

3. A least relative change Quasi-Newton update. We aim to solve the following constrained minimisation problem. Given A find B which satisfies (2.12). To derive the new update we will use the lemma used by Dennis and Schnabel to derive both the Broyden and Schubert updates.

The $n\times n$ matrix problem in the Frobenius norm (2.1) is identical to n vector minimisation problems in the 2-norm, i.e.,

$$\|B-A\|_F^2=\sum_{i=1}^{n}\|B_{i\bullet}-A_{i\bullet}\|_2^2 \tag{3.1}$$

where $A_{i\bullet}$ is the (column) vector obtained from row i of A and similarly for B. In the same way, problem (2.13) may be reformulated as

$$\sum_{i=1}^{n}\min\left[\sum_{j=1}^{n}\left(\frac{B_{ij}-A_{ij}}{A_{ij}}\right)^2\right]^{1/2} \tag{3.2}$$

and we can solve n problems by choosing a $B_{i\bullet}$ to solve for each element i of (3.2), i.e.,

$$\min\|W_i^+(B_{i\bullet}-A_{i\bullet})^T\|_2 \quad \text{subject to } B_{i\bullet}^Ts=y_i, \tag{3.3}$$

where $W_i=\operatorname{diag}(A_{i\bullet})$, i.e., a diagonal matrix formed from each row of A, and W_i^+ is its pseudoinverse.

In the following theorem we solve the minimisation problem for each row of the updated matrix. Since A is sparse W_i may have zero diagonal elements and its inverse will be not defined. In order to solve the minimisation problem we must use the pseudo inverse of W_i^+, defined in Theorem 3.6.5. in Dennis and Schnabel [10]. Since W_i^+ may itself also be singular we must define a further matrix $W_i^{\ddagger}$ in which all zero diagonal elements of W_i^{+} are replaced by a finite ε. The inverse of $W_i^{\ddagger}$ is then defined but in the limit as ε approaches zero the corresponding elements of $W_i^{\ddagger}(B_{i\bullet}-A_{i\bullet})$ will be zero and hence will not affect the minimisation.

THEOREM 1. *Let* $B_{i\bullet} \in L(R^n)$; $s, y \in R^n$; $s \neq 0$; *and let* W_i^{+} *be the pseudoinverse of* W_i; *then the solution to*

$$\min \| W_i^{+}(B_{i\bullet}-A_{i\bullet})^T \|_2 \quad \textit{subject to } B_{i\bullet}^T s = y_i \tag{3.3}$$

is

$$B_{i\bullet} = A_{i\bullet} + \frac{(y-As)_i s^T (W_i)^2}{\langle W_i s \cdot W_i s \rangle}. \tag{3.4}$$

Proof. First, consider the following problem, similar to (3.3):

$$\min \| W_i^{\ddagger}(B_{i\bullet}-A_{i\bullet}) \|_2 \quad \text{subject to } B_{i\bullet}^T s = y_i. \tag{3.5}$$

If we consider the matrix W_i^{+} as a scaling for each row of A, and $W_i^{\ddagger}$ is identical to W_i^{+} except that zero diagonal elements of W_i^{+} are replaced by ε, then the following scalings are equivalent as ε approaches zero

$$\tilde{B}_{i\bullet} = W_i^{+} B_{i\bullet} = W_i^{\ddagger} B_{i\bullet}, \qquad \tilde{A}_{i\bullet} = W_i^{+} A_{i\bullet} = W_i^{\ddagger} A_{i\bullet},$$

and

$$\tilde{s}_i = (W_i^{\ddagger})^{-1} s.$$

Equation (3.5) may be rewritten as

$$\min \| \tilde{B}_{i\bullet} - \tilde{A}_{i\bullet} \|_2 \quad \text{subject to } \tilde{B}_{i\bullet}^T \tilde{s}_i = y_i.$$

This is the usual quasi-Newton problem and, following the reasoning of Theorem 11.2.1 of Dennis and Schnabel [10], the solution is given by equation (3.4). In the limit as ε approaches zero (3.5) and (3.3) are identical since such elements will not contribute to $W_i^{\ddagger}(B_{i\bullet}-A_{i\bullet})$, and the solution is defined provided not all elements of s or W_i are zero. □

By combining the solutions of (3.3) for each row of B, we obtain the following update:

$$B = A + \sum_{i=1}^{n} e_i \frac{(y-As)_i s^T (W_i)^2}{s^T (W_i)^2 s} \tag{3.6}$$

where each matrix in the summation of the update corresponds to one row update of the form of equation (3.4). Like Schubert's update, each row of the update has a special form due, in this case, to its scaling by W_i^{+}. Each row has a unique denominator $s^T(W_i)^2 s$ and hence the update is best performed by rows.

Note that, unlike Schubert's method, it is not necessary that $B \in SP(Z) \cap Q$ but only $B \in Q$. By taking the pseudoinverse of W_i in which the inverse of zero elements are set to zero, we have taken into account the sparsity of the problem.

Equation (3.6) may also be written element by element as

$$B_{ij} = A_{ij} + \frac{(y-As)_i s_i (A_{ij})^2}{\sum_{k=1}^{n} s_k^2 (A_{ik})^2}. \tag{3.7}$$

If any A_{ij} is zero so also is B_{ij}. Hence any identically zero elements will remain zero and need not be stored. It is therefore wise to begin with an approximation with the same sparsity pattern as the Jacobian since if an arbitrary matrix were to be used elements initially assumed to be zero would remain zero. In this work a forward difference approximation to the Jacobian at the starting point has been used.

4. Scale invariance. A useful property held by the new update that is not shared by either Broyden's or Schubert's method is that of scale invariance. If D_1 and D_2 are nonsingular diagonal matrices, let the following scalings apply:

$$\hat{x} = D_1 x, \tag{4.1}$$

$$\hat{f} = D_2 f, \tag{4.2}$$

$$\hat{A} = D_2 A D_1^{-1}. \tag{4.3}$$

If $\{x_k\}$ is the sequence of vectors generated by a method for the unscaled problem and if $\{\hat{x}_k\}$ denotes the sequence generated for the problem scaled using (4.1) and (4.2), then the method is scale invariant if

$$\hat{x}_k = D_1 x_k \quad \forall k. \tag{4.4}$$

Paloschi and Perkins [16] set out the conditions under which a quasi-Newton method is scale-invariant. The following lemma gives the sufficient conditions for a quasi-Newton method being scale-invariant.

LEMMA [16]. *Given the changes of scale* (4.1) *and* (4.2) *and sequence of nonsingular matrices* $\{B_k\}$ *satisfying the following relation*:

$$\hat{B} = D_2 B D_1^{-1} \tag{4.5}$$

then a quasi-Newton method defined by

$$Ap = -f, \tag{4.6}$$

$$x_+ = x + p, \tag{4.7}$$

and

$$Bp = y \tag{4.8}$$

satisfies (4.4), *i.e., is scale-invariant.*

They demonstrated that a class of rank one quasi-Newton updates that are also scale invariant does exist. Bogle and Perkins [2] have shown that these results can be extended to obtain a similar class of sparse quasi-Newton updates that are not rank one. We will now show that the new update is also scale-invariant. The following scalings must also apply:

$$\hat{s} = D_1 s, \tag{4.9}$$

$$\hat{y} = D_2 y. \tag{4.10}$$

Let d_{1i} be the ith element of D_1, and similarly d_{2i} of D_2.

THEOREM 2. *Consider the method defined by the relations* (3.7), (4.6), *and* (4.7) *that generate the sequences* $\{x_k\}$ *and* $\{B_k\}$. *If for any change of scale of the form* (4.1) *and* (4.2) A_0 *is such that*

$$\hat{A}_0 = D_2 A_0 D_1^{-1}, \tag{4.11}$$

then the sequence $\{x_k\}$ *satisfies* (4.4), *i.e., the method is scale-invariant.*

Proof. Since A_0 satisfies (4.5), we need only show that if A satisfies (4.5) then so also does B. Then, by induction, so will the entire sequence $\{B_k\}$. Substituting the scalings in element by element it follows that

$$\hat{B}_{ij} = \frac{d_{21}A_{ij}}{d_{1j}} + \frac{\left(d_{2i}y_i - \sum_{k=1}^{n} \frac{d_{2i}A_{ik}}{d_{1k}} s_k d_{1k}\right) s_j d_{1j} \left(\frac{d_{2i}A_{ij}}{d_{1j}}\right)^2}{\sum_{k=1}^{n} (s_k d_{1k})^2 \left(\frac{d_{2i}A_{ik}}{d_{1k}}\right)^2} \tag{4.12}$$

$$= \frac{d_{21}A_{ij}}{d_{1j}} + \left(\frac{d_{21}}{d_{1j}}\right) \frac{(y - As)_i s_j (A_{ij})^2}{\sum_{k=1}^{n} s_k^2 (A_{ik})^2}, \tag{4.13}$$

and therefore B satisfies (4.5) for the entire sequence $\{B_k\}$. □

Neither Broyden's method nor Schubert's method is invariant to the scaling defined by (4.1) [1], [16].

5. Implementation. The three methods have been implemented in Fortran. The tests were done on a CDC Cyber 174 in single precision, which has a word length of 60 bits. In order to compare the methods, the same implementation has been used where possible. The only point where the implementations differ is in the solution of the set of linear equations. For Broyden's method we have used a standard LU decomposition with partial pivoting and updated using Bennett's algorithm [3]. This was found to give the same results on this machine as using a QR decomposition. For Schubert's method and the new method, the sparse matrix package MA28 (Duff [11]) has been used in which a sparse LU decomposition is obtain using the Markowitz pivoting strategy [12] and the matrix refactorised at each iteration using the pivot strategy obtained on the first iteration.

The method was said to have converged if the following test was satisfied within 200 iterations:

$$|f(x)| < 10^{-8}, \qquad i = 1, \cdots, n. \tag{5.1}$$

The results presented in Tables A2–A4 in the Appendix indicate the number of iterations required to solve the problem including those required to initialise the Jacobian by finite differences. Those problems for which a solution was not obtained are indicated by +. Since the problems are sparse, the method of Curtis, Powell, and Reid [8] has been used to economise on function evaluations for initialization. The number of independent differencing steps required for each problem is given in Table A1.

It has been found to be profitable when using quasi-Newton methods to reinitialise the Jacobian by differences if the method is making poor progress [16], [2]. Reinitialisation will be more important with the new method since if an element that is not an analytically zero element becomes zero, it will remain zero, which may cause a deterioration in progress until reinitialisation is invoked. In our code we have employed the following strategy. If $\|f_k\|_2$ is not reduced in 10 successive iterations, reinitialisation is performed at the point found so far with the best value of $\|f_k\|_2$.

A trust region method [14] has been implemented to maintain control of the stepsize. The step is forced to remain within a domain in each dimension i, δ_i, for the kth iteration defined by

$$\delta_{i0} = \max(50, 50|\xi_{i0}|),$$

$$\delta_{ik} = \begin{cases} 50, & \xi_{ik} = 0, \\ 50|\xi_{ik}|, & \xi_{ik} \neq 0, \end{cases}$$

where ξ_{ik} is the ith element of x_k.

6. Numerical results and conclusions. The new method (NEWQN) was tested along with Broyden's (BROYD) and Schubert's (SCHUB) methods on the eight problems listed in Table A1 of the Appendix each with 50 variables. Most are adaptations of standard test problems [15]. Problems 4 and 5, however, are new and result from the solution of equations resulting from a set of countercurrent reactors. The equations for problems 4, 5, and 6 are given in the Appendix. Each problem was attempted from its standard initial point x_0, from $10x_0$, and from $100x_0$ in order to test the robustness of each method, i.e., its ability to solve problems from a wide range of initial points. The methods have only been tested on relatively small sized problems (50 variables). This was done in order to make direct comparison between the new method and Schubert's method, which are sparse methods, and Broyden's method, which is a full matrix method, on an identical problem set. With identical implementations it is then possible to compare the effectiveness of the updates alone in their attempts to find a solution.

The results for the three methods are presented in the Appendix. Three indices have been used to collect the data for all 24 problems together for comparison. If r_{ij} is the number of iterations required to solve problem i with method j, r_{ib} the best result for problem i with any of the m methods (i.e., $r_{ib} = \min_j (r_{ij})$), t_j the number of successes by method j, and n_j the number of problems attempted by method j then we define the following indices:

Robustness index:

$$R_j = \frac{t_j}{n_j}. \tag{6.1}$$

Efficiency index:

$$E_j = \sum_{i=1,\, r_{ij}\neq 0}^{m} \left(\frac{r_{ib}}{r_{ij}}\right) \Big/ t_j. \tag{6.2}$$

Combined robustness and efficiency index:

$$E_j \times R_j = \sum_{i=1,\, r_{ij}\neq 0}^{m} \left(\frac{r_{ib}}{r_{ij}}\right) \Big/ n_j. \tag{6.3}$$

The summations are taken only over successful results. R is clearly a percentage of cases for which each method found a solution. For E and $E \times R$ indices larger values indicate a better result with 1.0 being the best result possible.

The results in Table 1 confirm the superior robustness of Broyden's method over Schubert's method, although when successful Schubert is the more efficient of the two. On this set of problems, the new method has a robustness equal to that of Broyden's method. Also, the method is more efficient than either of the other two methods. The results demonstrate that the new method is a significant improvement on Schubert's

TABLE 1
Summary of results.

	BROYD	SCHUB	NEWQN
Successes	22	18	22
R index	.92	.75	.92
E index	.654	.834	.899
$E \times R$ index	.600	.625	.824

TABLE A1
Problem set (50 variables unless otherwise indicated) with number of iterations required for initialisation by finite differences.

1	Discrete boundary value function	3
2	Broyden tridiagonal function	3
3	Broyden banded function	7
4	Counter current reactors problem 1	4
5	Counter current reactors problem 2	5
6	Brown almost linear function (tridiagonal version)	49
7	Extended Rosenbrock function	2
8	Extended Powell singular function (52 variables)	4

TABLE A2
Results from initial guess x_0 (number of iterations).

	BROYD	SCHUB	NEWQN
1	7	7	6
2	15	13	10
3	27	28	23
4	33	25	35
5	42	27	62
6	65	+	59
7	6	6	6
8	63	46	23

TABLE A3
Results from initial guess $10x_0$.

	BROYD	SCHUB	NEWQN
1	9	8	8
2	64	53	16
3	140	47	65
4	136	+	29
5	231	+	63
6	66	123	133
7	6	6	6
8	99	42	28

TABLE A4
Results from initial guess $100x_0$.

	BROYD	SCHUB	NEWQN
1	35	18	16
2	+	+	21
3	+	57	77
4	203	+	50
5	86	+	+
6	82	124	123
7	8	7	7
8	104	54	+

method for solving large sparse problems, although further testing will indicate to what extent this is true of broader classes of problems.

Appendix.

PROBLEM 4. Countercurrent reactors problem (1).

$$f_i(x) = \alpha x_{i-2} - (1-\alpha)x_{i+2} - x_i - \theta x_i x_{i+1}, \qquad i = 1, \cdots, 2n-1,$$
$$f_{i+1}(x) = \alpha x_{i-1} - (1-\alpha)x_{i+3} - x_{i+3} - x_{i+1} - \theta x_i x_{i+1},$$
$$x_{-1} = x_{n+2} = 1.0, \quad x_0 = x_{n+1} = 0.0, \quad \theta = 4.0, \quad \alpha = 0.5.$$

Initial point:

$$\underset{\sim}{x}_0 = (0.1, 0.2, 0.3, 0.4, 0.5, 0.4, 0.3, 0.2, 0.1, 0.2, \cdots).$$

PROBLEM 5. Countercurrent reactors problem (2).

$$f_1(x) = A_0 x_1 - (1-x_1)x_3 - A_1 - \theta A_1 x_2,$$
$$f_2(x) = B_0 x_1 - (1-x_1)x_4 - A_1 - \theta A_1 x_2,$$
$$f_3(x) = A_1 x_1 - (1-x_1)x_5 - x_3 - \theta x_3 x_4,$$
$$f_{i+1}(x) = x_1 x_{i-1} - (1-x_1)x_{i+3} - x_{i+1} - \theta x_i x_{i+1}, \qquad i = 4, \cdots, n,$$
$$A_0 = x_{n+2} = 1.0, \quad B_0 = x_{n+1} = 0.0, \quad \theta = 4.0, \quad A_1 = 0.414214.$$

Initial point:

$$\underset{\sim}{x}_0 = (0.1, 0.2, 0.3, 0.4, 0.5, 0.4, 0.3, 0.2, 0.1, 0.2, \cdots).$$

PROBLEM 6. Brown almost linear function (modified).

$$f_1(x) = 2x_1 + x_2 - 3,$$
$$f_i(x) = x_{i-1} + 2x_i + x_{i+1} - 4, \qquad i = 2, \cdots, n-1,$$
$$f_n(x) = \left(\frac{\sum_{i=1}^{10} \prod_{j=1}^{n/10} (x_{n(i-1)+10j}/10)}{10} \right) - 1.0.$$

Initial point:

$$\underset{\sim}{x}_0 = 0.5.$$

Acknowledgments. The authors would like to thank the reviewers for their helpful comments, which assisted in clarifying a number of points.

REFERENCES

[1] I. D. L. BOGLE, *The numerical solution of flowsheeting problems characterised by large sparse Jacobian matrices*, Ph.D. Thesis, University of London, London, UK, 1983.

[2] I. D. L. BOGLE AND J. D. PERKINS, *Sparse Newton-like methods in equation oriented flowsheeting*, Comput. Chem. Engrg., 12 (1988), pp. 791-805.

[3] J. M. BENNETT, *Triangular factors of modified matrices*, Numer. Math., 7 (1965), pp. 217-221.

[4] C. G. BROYDEN, *A class of methods for solving nonlinear simultaneous equations*, Math. Comp., 19 (1965), pp. 577-593.

[5] ———, *The convergence of an algorithm for solving sparse nonlinear systems*, Math. Comp., 25 (1971), pp. 285-294.

[6] P. H. CALAMAI AND J. J. MORÉ, *Quasi-Newton methods with bounds*, Tech. Report ANL/MCS-TM-60, Argonne National Laboratory, Argonne, IL, 1987.

[7] H. CHEN AND M. STADTHERR, *A sparse nonlinear equation solver for process flowsheeting*, AIChE Meeting, Los Angeles, CA, November 1982.

[8] A. CURTIS, M. J. D. POWELL, AND J. K. REID, *On the estimation of sparse Jacobian matrices*, J.I.M.A. 13 (1974), pp. 117-120.

[9] J. E. DENNIS AND R. B. SCHNABEL, *Least change secant updating for quasi-Newton methods*, SIAM Rev., 21 (1979), pp. 443-459.

[10] ———, *Numerical Methods for Unconstrained Optimisation and Nonlinear Equations*, Prentice Hall, Englewood Cliffs, NJ, 1983.

[11] I. S. DUFF, MA28—*A set of* FORTRAN *subroutines for sparse unsymmetric linear equations*, UKAEA Harwell Tech. Report AERE-R8730, H.M.S.O., AERE Harwell, London, UK, 1979.

[12] H. M. MARKOWITZ, *The elimination form of the inverse and its application to linear programming*, Management Sci., 3 (1957), pp. 225-269.

[13] E. MARWIL, *Convergence results for Schubert's method for solving sparse nonlinear equations*, SIAM J. Numer. Anal., 16 (1979), pp. 588-604.

[14] J. J. MORE, *Recent developments in algorithms and software for trust region methods*, in Mathematical Programming: The State of the Art, A. Bachem, M. Grotschel, and B. Korte, eds., Springer-Verlag, Berlin, New York, 1983, pp. 258-287.

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

[16] J. R. PALOSCHI AND J. D. PERKINS, *Scale invariant quasi-Newton methods for the solution of nonlinear equations*, Comput. Chem. Engrg., 12 (1987), pp. 91-97.

[17] J. D. PERKINS AND R. W. H. SARGENT, SPEEDUP: *A computer program for steady state and dynamic simulation and design of chemical processes*, AIChE Symposium Ser., 78 (1982), pp. 1-11.

[18] L. K. SCHUBERT, *Modification of a quasi-Newton method for nonlinear equations with a sparse Jacobian*, Math. Comp., 24 (1970), pp. 27-30.

[19] Ph. L. TOINT, *Numerical solution of large sets of nonlinear equations*, Math. Comp., 46 (1986), pp. 175-189.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 4, pp. 631-647, July 1990

A FOURIER-SERIES METHOD FOR SOLVING SOLITON PROBLEMS*

C. I. CHRISTOV† AND K. L. BEKYAROV‡

Abstract. A Fourier-Galerkin method with an earlier proposed complete orthonormal system of functions in $L^2(-\infty, \infty)$ as the set of trial functions is developed and displayed for the problem of calculating the shape of the one-soliton solution of the Korteweg-de Vries equation. The convergence of the method is investigated through comparison with the analytic solution, which appears to be very good. The truncation and discretization errors are assessed pointwise. The technique developed is also applied to the soliton problem for the so-called Kuramoto-Sivashinsky equation and the obtained soliton shape is compared to the existing difference solution. The quantitative agreement between the Fourier-series-method result and the numerical one is good. In the present paper, however, the soliton solution is obtained for a significantly wider range of phase velocities, which suggests that the spectrum might be continuous. The new technique can also be applied to a variety of other problems involving identification of homoclinic solutions.

Key words. solitons, spectral methods, Fourier-Galerkin method, Korteweg-de Vries, Kuramoto-Sivashinsky

Introduction. In recent years, the problem of calculating shapes of solitons has attracted considerable attention due to its application in different fields of modern physics (see, e.g., [1], [2]). At this time the available techniques for calculating solitons lack generality and, as a rule, bear semianalytical character. Each of these techniques proves effective only for the particular class of equations for which it is devised. In turn the numerical approaches based on straightforward difference approximations are faced with formidable challenges. The first one is rooted in the inverse nature of the boundary value problem in an unbounded region, forcing us to employ shooting procedures that are intrinsically highly unstable. One of the effective means of overcoming that difficulty appears to be the method of variational imbedding [3], [4] based on rendering the original inverse problem to a higher-order but correct boundary value problem. The second challenge is connected with the choice of the "actual" infinity for the difference approximations and often shows itself through the occurrence of artificial eigenvalue problems that are not characteristic for the original problem in an unbounded region.

A method free from said shortcoming is that of Fourier-series expansion with respect to certain complete orthonormal (CON) system of functions in $L^2(-\infty, \infty)$ space. However, the governing equations for solitons are, as a rule, nonlinear. For the employed CON system, this places the very stringent requirement of possessing, for the product of two members of the system, a representation in series with respect to the system. It should be noted that for the well-known sets of Hermitian functions in $L^2(-\infty, \infty)$ and Laguerre functions in $L^2[0, \infty)$, such a representation is not available (see, e.g., [5]).

The most systematic way to devise a CON system with the required properties is, perhaps, to map through an algebraic function the infinite interval into $[-1, 1]$ and then to use the CON set of Chebyshev polynomials. This idea was initially sketched

* Received by the editors August 22, 1988; accepted for publication (in revised form) May 31, 1989. This research was supported in part by the Ministry of Culture Science and Education of Bulgaria under grant 1052.

† Laboratory of Hydrometeorological Informatics, Hydrometeorological Service, Bulgarian Academy of Sciences, boul. Lenin 66, Sofia 1184, Bulgaria.

‡ Department of Atmosphere Physics, Institute of Geophysics, Bulgarian Academy of Science, "Akad. G. Bonchev" str. block 3, Sofia 1113, Bulgaria.

by Grosch and Orszag [6] for the semi-infinite interval and was generalized by Boyd [7], who gave the appropriate mapping for the entire interval $(-\infty, \infty)$ map and built the rigorous basis beneath the said technique. Boyd [8] coined the term "rational Chebyshev functions" for the newly devised set of functions (which we prefer to call Boyd functions in order to stress the specifics in the unbounded intervals) and presented a scrupulous analysis of the convergence region of the series with respect to them.

Another system suitable for nonlinear problems in infinite intervals is proposed in the earlier work [9] and developed further in [10]. Unlike the above-mentioned works of Boyd, in the works of Christov and Bekyarov the emphasis is put on the performance of the spectral method in the infinite interval for intrinsically nonlinear problems.

In the present paper the numerical technique for application of the Fourier method based on the CON system from [9] is developed. The method is featured through solving the soliton problem for the Korteweg-de Vries (KdV) equation and for the equation of weakly nonlinear approximation in falling down a wall of thin viscous capillary films (sometimes called the Kuramoto-Sivashinsky equation). Although some authors use the term "soliton" only for special kinds of solitary waves, in the present work this term is used as a synonym for the term "solitary wave."

1. Posing the problem. For the sake of simplicity consider the following form of the Korteweg-de Vries equation (see [2, Chap. 3]):

$$u_t - 6uu_x + u_{xxx} = 0, \tag{1.1}$$

and seek a solution of the type of propagating wave $u = u(\xi)$, where $\xi = x - at$, and $a > 0$ is the phase velocity of the wave. Then (1.1) reduces to

$$-au' - 6uu' + u''' = 0,$$

where the prime stands for a differentiation with respect to the independent variable ξ. We have a soliton solution (a solitary wave) if the following boundary conditions hold:

$$u(\xi) \to 0 \quad \text{for } \xi \to \pm\infty. \tag{1.2}$$

Under these boundary conditions the above ordinary differential equation can be integrated once and rendered to

$$-au - 3u^2 + u'' = 0. \tag{1.3}$$

Thus (1.3) and (1.2) form the boundary value problem to be solved. Fortunately, the latter possesses an analytical solution for each $a > 0$:

$$u = -\frac{a}{2}\operatorname{sech}^2[\sqrt{a/2}\,\xi] \tag{1.4}$$

that can be used for checking the accuracy of the numerical schemes proposed.

Another interesting one-dimensional nonlinear equation of evolution arises in the weakly nonlinear approximation for the shape of the free surface of thin film of viscous liquid falling down a vertical plane when the capillary forces are significant. The rich phenomenology of this flow made it one of the most popular and spurred a formidable amount of research papers. It is not the purpose of the present work to go into the details connected with that flow and we refer the reader to [11] and [12] for a comprehensive review of the experimental and theoretical approaches, respectively. For our purposes it is enough to cite here that in the frame of the weakly nonlinear approximation [13] the following dimensionless equation for evolution of the scaled film thickness φ in a moving frame can be derived [14], [15]:

$$\frac{\partial \varphi}{\partial t}+6\varphi\frac{\partial \varphi}{\partial x}+\frac{\partial^2 \varphi}{\partial x^2}+\frac{\partial^4 \varphi}{\partial x^4}=0. \tag{1.5}$$

The same equation is arrived at in [16] when modeling chemically reacting fronts and flames. Now (1.5) is called the Kuramoto-Sivashinsky equation (for brevity the K-S equation). A thorough investigation of the different regimes is given in [17], where the bifurcation catalogue of the attractors of the K-S equation is compiled on the basis of several hundred numerical experiments.

Consider once again a solution of the type of propagating wave $\varphi=\varphi(\xi)$, $\xi=x-ct$ when (1.5) is rewritten as follows:

$$-c\varphi'+6\varphi\varphi'+\varphi''+\varphi^{(4)}=0$$

and when under the boundary conditions for soliton solutions

$$\varphi(\xi)\to 0 \quad \text{for} \quad \xi\to\pm\infty \tag{1.6}$$

we can integrate once and obtain

$$-c\varphi+3\varphi^2+\varphi'+\varphi'''=0. \tag{1.7}$$

Unlike for the Korteweg-de Vries equation for the last boundary value problem (1.6), (1.7) an analytic solution is not yet available. Rather, different numerical techniques are applied [4], [12], [18], [19] and the shape of the one-soliton solution is calculated fairly reliably, but the question of which is the type of spectrum (continuous or discrete) for the eigenvalue parameter c is still unresolved. Having the Fourier method verified in the case of the KdV equation, we apply it to boundary value problem (1.6), (1.7) and the soliton solution obtained compares very well with the finite-difference solution [4]. The interesting finding here supporting the difference-method results of [4] is that we were able to find the approximate solutions for the soliton problem for each value of celerity c, which we tried up to 20. This allows us to state the hypothesis that the spectrum for which solitons of the K-S equation exist is continuous $0<c<\infty$.

It should be mentioned that we consider only smooth classical solutions to (1.3) or (1.7), which yields that under conditions (1.2) and (1.6) these solutions should belong to the $L^2(-\infty,\infty)$ space. So we occupy ourselves in what follows with solutions for which the following requirement holds:

$$\int_{-\infty}^{\infty}\varphi^2(\xi)\,d\xi<+\infty \quad \text{or} \quad \int_{-\infty}^{\infty}u^2(\xi)\,d\xi<+\infty. \tag{1.8}$$

2. The CON system. Wiener [20, p. 35] introduced the system

$$\rho_n=\frac{1}{\sqrt{\pi}}\frac{(ix-1)^n}{(ix+1)^{n+1}}, \qquad n=0,1,2,\cdots \tag{2.1}$$

as a Fourier transform of Laguerre functions. Higgins [21] defined it also for negative n and proved its completeness and orthogonality. The significance of (2.1) for nonlinear problems is revealed in [9], where the product formula is derived:

$$\rho_n\rho_k=\frac{i}{2\sqrt{\pi}}(\rho_{n+k}-\rho_{n-k}) \tag{2.2}$$

and the two real subsequences of odd functions S_n and even functions C_n are introduced according to the formulae

$$S_n = \frac{(\rho_n + \rho_{-n-1})}{i\sqrt{2}}, \quad C_n = \frac{(\rho_n - \rho_{-n-1})}{\sqrt{2}}, \quad n = 0, \pm 1, \cdots . \tag{2.3}$$

The sequence can be reduced just to its portion with positive values of index n due to the symmetry property:

$$S_{-n} = S_{n-1}, \qquad C_{-n} = -C_{n-1}. \tag{2.4}$$

The explicit expressions for S_n and C_n can be found in [9]. A simple representation in terms of trigonometric functions has been brought to our attention by Geshev [22]:

$$\begin{aligned} S_n(x) &= (-1)^{n+1}\frac{\sin(n+1)\vartheta + \sin n\vartheta}{\sqrt{2}}, \\ C_n(x) &= (-1)^{n}\frac{\cos(n+1)\vartheta + \cos n\vartheta}{\sqrt{2}}, \end{aligned} \tag{2.5}$$

where $\vartheta = 2\,\mathrm{arctg}\,(x)$. These and other formulae interrelating the CON system employed here to the different families of Boyd's functions are presented in [23].

Most of the practically important formulae for the system (2.3) are compiled in [9] and [10], and here we cite only those that are necessary for carrying out the present calculations.

The most important feature of the system, namely, equality (2.2) for the real-valued subsequences S_n, C_n adopt the form

$$\begin{aligned} S_nS_m &= \sum_{k=0}^{\infty} \alpha_{nmk}C_k, \\ \alpha_{nmk} &= \frac{1}{2\sqrt{2\pi}}\left\{\delta_{k,n+m+1} - \delta_{k,n+m} + \delta_{k,|n-m|} - \mathrm{sgn}\left[|n-m| - \frac{1}{2}\right]\delta_{k,[|n-m|-1/2]}\right\}, \end{aligned} \tag{2.6.SS}$$

where $\delta_{i,j}$ is Kronecker delta and $[\cdot]$ stands for the integer part of a real. Respectively,

$$\begin{aligned} C_nC_m &= \sum_{k=0}^{\infty} \beta_{nmk}C_k, \\ \beta_{nmk} &= \frac{1}{2\sqrt{2\pi}}\left\{-\delta_{k,n+m+1} + \delta_{k,n+m} + \delta_{k,|n-m|} - \mathrm{sgn}\left[|n-m| - \frac{1}{2}\right]\delta_{k,[|n-m|-1/2]}\right\}, \end{aligned} \tag{2.6.CC}$$

and

$$\begin{aligned} S_nC_m &= \sum_{k=0}^{\infty} \gamma_{nmk}S_k, \\ \gamma_{nmk} &= \frac{1}{2\sqrt{2\pi}}\{-\delta_{k,n+m+1} + \delta_{k,n+m} + \mathrm{sgn}\,(n-m)\delta_{k,|n-m|} - \mathrm{sgn}\,(n-m)\delta_{k,|n-m|-1}\}. \end{aligned} \tag{2.7.SC}$$

In the same manner the formulae representing derivatives of a member of the system into series with respect to the system are derived:

$$C_n' = \sum_{m=0}^{\infty} \theta_{nm}S_m, \qquad S_n' = -\sum_{m=0}^{\infty} \theta_{nm}C_m, \tag{2.8a}$$

where

$$\theta_{mn}=\frac{n}{2}[\delta_{m,n}-\delta_{m,n-1}]-\frac{n+1}{2}[\delta_{m,n+1}-\delta_{m,n}].$$

(2.8b)
$$C''_n=\sum_{m=0}^{\infty}\chi_{nm}C_m,\qquad S''_n=\sum_{m=0}^{\infty}\chi_{nm}S_m,$$

where

$$\chi_{nm}=-\frac{1}{4}n(n-1)\delta_{m,n-2}+n^2\delta_{m,n-1}-\frac{n^2+(2n+1)^2+(n+1)^2}{4}\delta_{n,m}$$
$$+(n+1)^2\delta_{m,n+1}-\frac{1}{4}(n+1)(n+2)\delta_{m,n+2}.$$

(2.8c)
$$C'''_n=\sum_{m=0}^{\infty}\varphi_{nm}S_m,\qquad S'''_n=-\sum_{m=0}^{\infty}\varphi_{nm}C_m,$$

where

$$\varphi_{nm}=\frac{1}{8}n(n-1)(n-2)\delta_{m,n-3}-\frac{3}{8}n(n-1)(2n-1)\delta_{m,n-2}$$
$$+\frac{3}{8}n(5n^2+1)\delta_{m,n-1}+\frac{3}{8}(n+1)[5(n+1)^2+1]\delta_{m,n+1}$$
$$-\frac{4n^3+4(n+2)^3+(2n+1)[n^2+(2n+1)^2+(n+1)^2]}{8}\delta_{m,n+1}$$
$$-\frac{3}{8}(n+1)(n+2)(2n+3)\delta_{m,n+2}+\frac{1}{8}(n+1)(n+2)(n+3)\delta_{m,n+3}.$$

3. Fourier series for the Korteweg–de Vries equation. In order to display the performance of the Fourier-series method for solving soliton problems, we begin with the Korteweg–de Vries equation. Since the sole purpose of the present work is to check the applicability of the new CON system to the case of intrinsically nonlinear problems, we are not concerned with the problem of which of the spectral techniques will work most efficiently: Galerkin, collocation, or tau version (see [24] for a comprehensive review of the spectral methods), and as so we do not create different algorithms to implement each of them for the purposes of comparison. Rather we choose the Galerkin scheme, because in our case it is cheaper in implementation due to the fact that the matrix is sparse. In fact, the part of the matrix that is responsible for the linear terms is a band matrix with seven nontrivial diagonals (see (2.8c)), whereas the part responsible for the nonlinear terms has only four nontrivial diagonals (see (2.6) and (2.7)) that are not adjacent, however. It is clear that for equations with more complicated nonlinear terms it is better to use the collocation (pseudospectral) technique, since the latter is much easier to program and the respective algorithms are easily verified. The only reason to decide against it in the present paper is that the pseudospectral method always converts the differential equation into a system of nonlinear algebraic equations with dense matrix. So we resort here to the Fourier–Galerkin method.

It is easily shown that (1.3) admits even functions as solutions and hence we develop the solution to be u into series only with respect to the even subsequence of functions C_n, namely,

(3.1)
$$u(x)=\sum_{n=0}^{\infty}a_nC_n(x).$$

Then for the terms entering (1.3), we obtain

$$u''(x)=\sum_{m=0}^{\infty}\sum_{n=0}^{\infty}a_m\chi_{mn}C_n(x) \tag{3.2}$$

and

$$u^2(x)=\sum_{m_1=0}^{\infty}\sum_{m_2=0}^{\infty}\sum_{n=0}^{\infty}a_{m_1}a_{m_2}\beta_{m_1m_2n}C_n(x). \tag{3.3}$$

Since for the Galerkin method the sets of trial and test functions coincide with the set C_n, then upon introducing (3.1)-(3.3) into (1.3), combining the terms with the like functions C_n, and taking the respective coefficients to be equal to zero (due to the independence of members of subsequence C_n and its completeness in the subspace of even $L^2(-\infty,\infty)$ functions), we obtain the following infinite nonlinear algebraic system for the unknown coefficients a_n:

$$-a\cdot a_n-3\sum_{m_1=0}^{\infty}\sum_{m_2=0}^{\infty}a_{m_1}a_{m_2}\beta_{m_1m_2n}+\sum_{m=0}^{\infty}a_m\chi_{mn}=0,\qquad n=0,1,2,\cdots. \tag{3.4}$$

For the sake of definiteness we take $a=1$, which does not cause a loss of generality, since substituting $u(x)=av(ax)$ we can exclude the parameter a from the equation.

It is obvious that we can solve only finite versions of (3.4) and the number N of unknown coefficients at which the vector $\mathbf{a}=\{a_n\}$ is truncated corresponds to the number of equations of (3.4) that are retained. The chief purpose of the present work is to show the efficiency of the Fourier-Galerkin method with the CON basis set from [9] for nonlinear problems, as is done in [23] for the case of linear equation with polynomial coefficients. The sole trait of efficiency of a spectral method is the capability to give good approximation with a sufficiently small number $(N+1)$ of functions, used in the series (see, e.g., [24, Chap. 1] and this will be the central issue of what follows.

For this reason we are not concerned here with the problems of the particular numerical implementation of the procedure for solving the nonlinear algebraic system representing the truncated version of (3.4). These problems require a detailed treatment if they are to be tackled and this goes far beyond the framework of the present paper. For now we need only a sufficiently rapid robust procedure for solving nonlinear systems. Moreover, here we solve only one-dimensional problems when the required computational time is small. We found satisfactory the pseudo-Newton's widely used algorithm of Brent (see [25]). The problem of efficiency of the numerical procedure shall, however, inevitably arise when multidimensional soliton problems yielding vast algebraic systems are to be considered.

The general consequence of the algorithm is as follows:

(i) We begin with the case $N=0$ when the system reduces to just one equation for the unknown a_0, which has two solutions

$$a_0^{(1)}=0 \quad\text{and}\quad a_0^{(2)}=-0.835543.$$

Here is shown the bifurcation character of the problem under consideration. In this simple case we are fortunate to solve the intricate problem for existence of a nontrivial solution at the stage $N=0$. This will be not the case for some other equations and, in general, we must try with increasing N in order to find a nontrivial solution.

(ii) Having obtained the solution for certain $N=M$, it is used as an initial condition for calculations with $N=M+1$ coupling it simply with the initial condition $a_{M+1}=0$. After the convergence of the numerical procedure of the Brent method is attained, the $(M+1)$th approximation is completed.

(iii) The calculations are terminated when the first $K+1$ unknowns a_i, $i=0,1,2,\cdots,K$ cease to change with increasing the number of equations, and more specifically, when the following criterion is satisfied:

$$(3.5)\qquad |a_i^{(M)}-a_i^{(M+1)}|<\varepsilon\|\mathbf{a}\|,\qquad i=0,1,\cdots,K,$$

where $\|\mathbf{a}\|=(a_0^2+\cdots+a_M^2)^{1/2}$ is the Euclidean norm of the solution and ε is a small quantity. In our calculations we took $\varepsilon=10^{-3}$, $K=6$ and the convergence in the said sense was obtained for $N=17$. Table 1 gives an insight into the manner in which the convergence is attained. It is seen that the convergence is very rapid and for $N=6$ even a_3 changes only with quantity of approximately 0.005.

In order to check the accuracy of the Fourier method, we used two different methods. The first consists of developing the exact solution (1.4) into a series with respect to the system using the Simpson formula with fourth-order approximation for evaluation of the respective integrals taken over the region $x\in[0,20]$ with 501 grid points. The coefficients obtained in this manner comprise the first column of Table 1. We can easily see the excellent agreement for the first 10 coefficients $a_0,\cdots,a_9$ of the exact solution, and calculated with $N=10$, the approximate solution. This is a certificate for good performance of the method proposed.

The second way of verifying the method is the comparison with the exact solution for the soliton shape itself. Such a comparison is depicted in Fig. 1, and the convergence for the shape is so rapid that even the approximate solution with $N=7$ cannot be discerned from the exact one. The most striking thing, however, is that the solution with $N=1$ (two terms in the truncated series) differs less than with 0.025 from the exact solution at the time when the size of solution is 0.5, i.e., the difference is within five percent from the maximal value. The latter means that in certain cases the present method can serve as a method for express assessment of the solution shape that requires solving just a couple of nonlinear algebraic equations.

It is interesting to assess the error of the Fourier-Galerkin method. The analytic solution (1.4) of the nonlinear KdV equation gives us the unique opportunity to

TABLE 1

Developing the solution by increasing the number of equations, and comparison with the respective coefficients obtained by developing the analytic solution into Fourier series.

	Analytic solution	$N=0$	$N=1$	$N=4$	$N=6$	$N=9$
a_0	−0.7925	−0.8355	−0.8101	−0.7994	−0.7930	−0.7925
a_1	−0.1760		−0.163	−0.1688	−0.1731	−0.1760
a_2	0.0190			0.0137	0.0203	0.0190
a_3	0.0588			0.0371	0.0567	0.0588
a_4	0.0500			0.0176	0.0444	0.0501
a_5	0.0319				0.0237	0.0321
a_6	0.0167				0.0078	0.0167
a_7	0.0067					0.0066
a_8	0.0009					0.0008
a_9	−0.0010					−0.0009
Maximal absolute error (3.8)		0.167	0.0430	0.0243	0.0046	0.0016

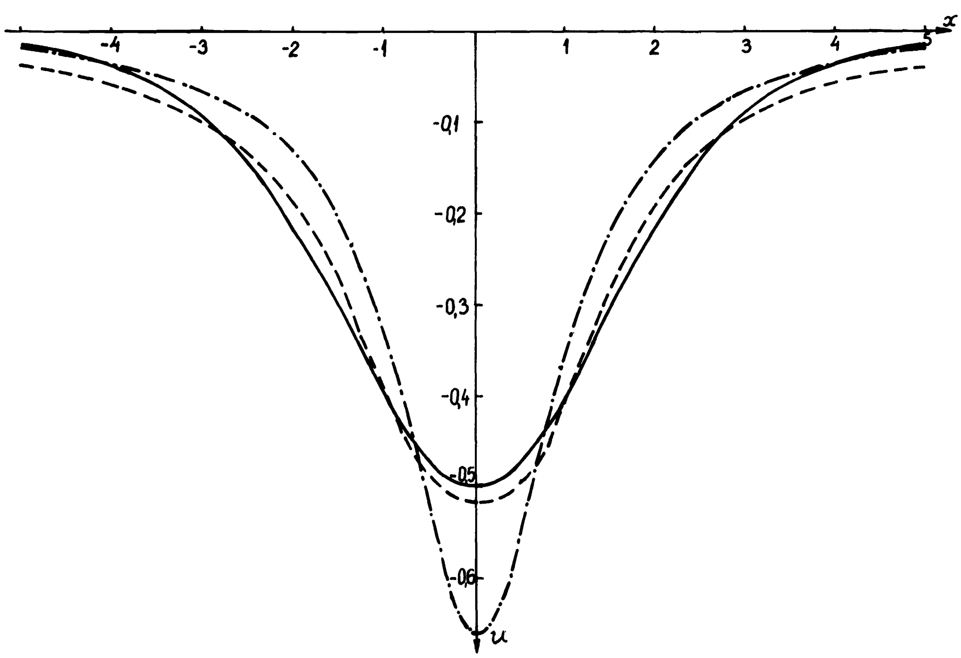


FIG. 1. *Developing the calculated shape for the solitons of the Korteweg-de Vries equation by increasing the number of modes* $N+1$: $-\cdot-$ $(N=0)$; $---$ $(N=1)$ *and comparison with the analytical solution*: ——.

quantitatively assess the part of the absolute error due to the truncation of the series—the truncation error (see, e.g., [8], [24] for definition):

$$E_T(x;\ N) \equiv u(x) - \sum_{n=0}^{N} (a_n S_n + \bar{a}_n C_n). \tag{3.6}$$

Taking a_n, $\bar{a}_n$ from the first column of Table 1 (i.e., the "true" ones) we obtain the pure contribution to the error due to the truncation of the series. This error as a function of the spatial coordinate x is depicted in Fig. 2.

Following [8] we also consider the so-called discretization error:

$$E_D(x;\ N) \equiv \sum_{n=0}^{N} [(a_n - a_n^{(N)})S_n + (\bar{a}_n - \bar{a}_n^{(N)})C_n], \tag{3.7}$$

where the quantities denoted by a superscript N are the solution of the truncated system, whereas those without a superscript are the coefficients of the series for the analytic solution (see the first column in Table 1). Discretization error as a function of x is shown in Fig. 3.

In most of the cases, an analytical solution is not available and hence the two kinds of errors above cannot be calculated explicitly. For this reason the absolute error (which is not a simple sum of the truncation and discretization errors) is the only one that can be assessed in the numerical computations:

$$E_\Delta(x;\ N) \equiv u(x) - \sum_{n=0}^{N} (a_n^{(N)} S_n + \bar{a}_n^{(N)} C_n), \qquad e_\Delta(N) = \max_x |E_\Delta(x;\ N)|, \tag{3.8}$$

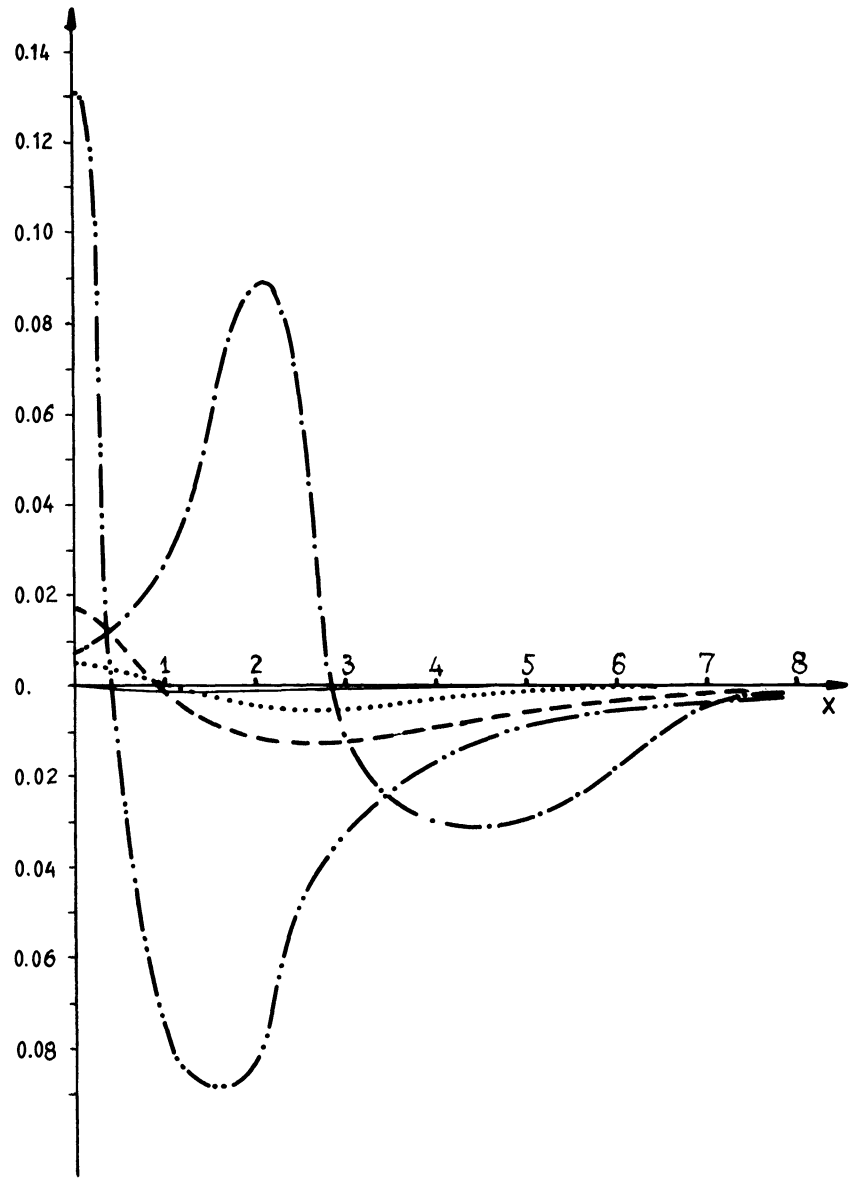


FIG. 2. *The truncation error for the Korteweg–de Vries equation as a function of x for different N*: – · · – ($N=0$); – · – ($N=1$); – – – ($N=4$); · · · · · ($N=6$); —— ($N=9$).

where $u(x)$ is either the analytic solution (when available) or the numerical spectral solution with certain sufficiently large number N_∞ of terms pertained, i.e.,

$$u(x)=\sum_{n=0}^{N_\infty}\left(a_n^{(N_\infty)}S_n+\bar{a}_n^{(N_\infty)}C_n\right). \tag{3.9}$$

The last row of Table 1 presents the maximal with respect to the x value $e_\Delta(N)$ of the absolute error. The rapid decrease of the absolute error with the increase of the number of the retained terms in the series is well seen. The latter is very important since the solution (1.4) decays exponentially at infinity, whereas the functions of the

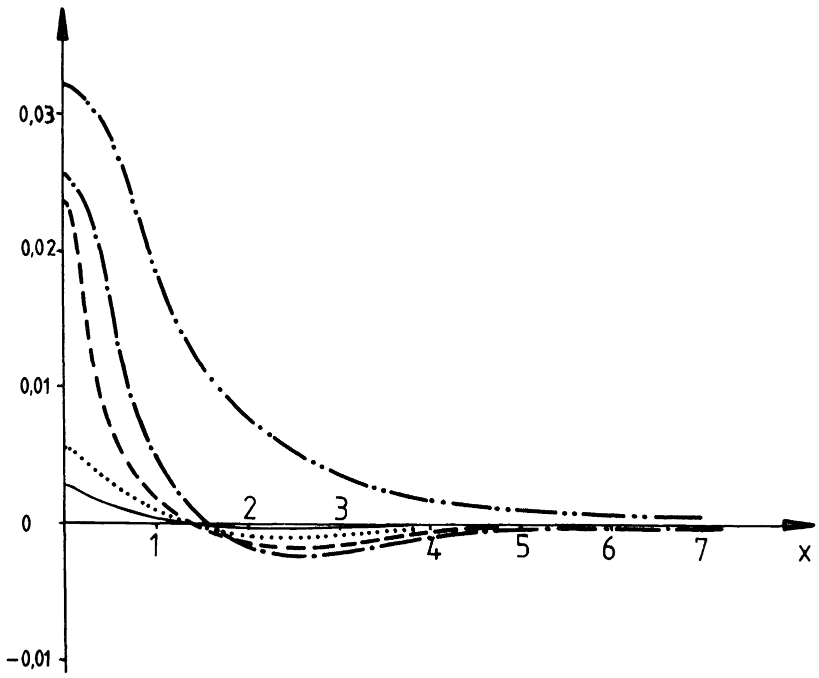


FIG. 3. *The discretization error for the Korteweg–de Vries equation as a function of x for different N*: –·····– ($N=0$); –·– ($N=1$); ––– ($N=4$); ······ ($N=6$); —— ($N=9$).

employed CON system decay as x^{-2}. There is no doubt that at $x \to \infty$ the approximate spectral solution poorly represents the real behavior of the solution. In that region, however, the magnitude of the solution is small and does not bear significant practical importance. That is, the method proposed is adequate enough for practical purposes, even in situations where the asymptotic behavior of the sought solution differs significantly from the respective behavior of the functions from the basis set.

4. Optimization of the method. The delightful results of the previous section are a matter of luck in a sense, because the characteristic measure of the support of the sought function turns out to be close to that of the employed set of functions, and more specifically for the first couple of members. If it happens that this is not the case and those two characteristic lengths are not close enough, a significantly greater number of terms might be needed to secure acceptable approximation.

Fortunately, the unboundedness of the considered interval always allows us to bring the mentioned characteristic lengths in correspondence, since if $F(x) \in L^2(-\infty, \infty)$ then $F(\beta x) \in L^2(\infty, \infty)$, where $\beta > 0$ is real. The idea to scale the independent variables is also employed in [6]–[8], and for the scale factor it is shown that its optimal value may even depend on the number of terms retained in the truncated spectral series. It is simpler to scale the independent variable prior to the calculations and thus render the characteristic length of the sought solution in accordance with the scales length of functions S_0, C_0. It is important to mention here that the adequate choice of β in [9] allowed us to reduce for the Burgers equation the required nontrivial coefficients a_i just to one: the coefficient a_0. That was possible since the soliton problem there reduced to a first-order ordinary differential equation. When higher-order equations are considered such a drastic reduction is not to be expected but still the solution can be significantly improved. The problem is to devise a quantitative criterion for discerning the better solutions. Qualitatively speaking, a solution is better when the higher-order coefficients represent a smaller share from the norm of the vector of coefficients a_i. One of the possible versions a criterion securing the presence of that property is the following:

$$I(\beta)=\sum_{n=0}^{N}|a_n(\beta)|n^2=\min. \tag{4.1}$$

In the last formula, the coefficients $a_i(\beta)$ are the solution of the system

$$-a\cdot a_n-3\sum_{m_1=0}^{N}\sum_{m_2=0}^{N}a_{m_1}a_{m_2}\beta_{m_1m_2n}+\frac{1}{\beta^2}\sum_{m=0}^{N}\chi_{mn}a_n=0, \tag{4.2}$$

$$n=0,1,2,\cdots.$$

The reason for seeking the value $\beta=\beta_{\min}$ for which the minimum of I is attained is that this can be done for a relatively small value of N and only after that to run the final calculations with higher N. The quest for minimum turns out to be a very inexpensive procedure, since for each new value β the solution for the previous one serves as an initial condition and the iterative procedure [25] converges rapidly.

The minimum of $I(\beta)$ is sought in the interval $0.5\leqq\beta\leqq 25$ by means of the method of the golden section [26]. For $N=5$ the sought value is $\beta_{\min}\approx 2.84$. Here we mention that a couple of different criteria have been checked, e.g.,

$$I(\beta)=\sum_{n=0}^{N}a_n^2(\beta)n^2=\min \tag{4.3}$$

or

$$I(\beta)=\sum_{n=0}^{N}a_n^2(\beta)n^4=\min, \tag{4.4}$$

and what is amazing is that the optimal scale factor is always $\beta_{\min}\approx 2.8$. Table 2 gives an insight into the way in which the solution for a_i depends on β for the case $N=4$. It is seen that a_2 promptly decreases near the optimal value of β. Figure 4 shows the solution calculated only on the basis of the first two coefficients a_0, a_1 with two different values of β. It is interesting to note that for the optimal value $\beta=2.8$, the solution virtually coincides with the analytic one and cannot be discerned in the figure.

Here we note that calculating the scale factor β in accordance with the adopted criterion does not necessarily yield for a fixed N a solution with least value for the

TABLE 2
The dependence of a_i on i for different scale factors β when $N=4$.

β	a_0	a_1	a_2	a_3	a_4	Maximal error, compared to $N=10$
1	−0.7994	−0.1688	0.0137	0.0371	0.0176	0.0243
2	−0.5640	0.1086	0.0568	0.0112	−0.0002	0.0033
2.8	−0.4513	0.1806	−0.0004	−0.0080	−0.0002	0.0021
4	−0.3475	0.2043	−0.0643	0.0088	−0.0015	0.0003
5	−0.3377	0.2136	−0.0587	0.0091	−0.0027	0.066

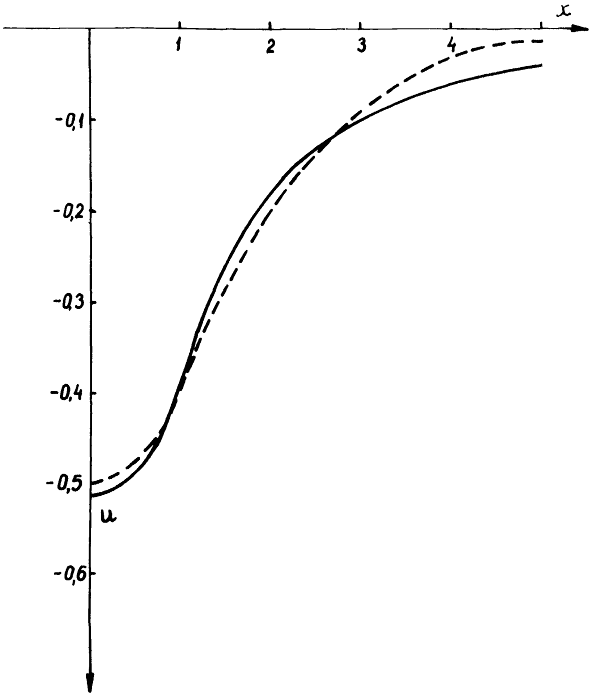


FIG. 4. *The shape of the soliton for the Korteweg-de Vries equation as given by the first two terms* a_0, a_1 *with different* β: --- ($\beta = 2.8$) *and analytic solution*; —— ($\beta = 1$).

maximal absolute error. This is easily seen in Table 2, where the least error is attained for $\beta = 4$. The purpose of the optimization proposed here is to minimize the number N of needed terms in series that are enough to give quantitatively adequate approximation.

5. Solitons for the Kuramoto–Sivashinsky equation. Deriving confidence from the successful attempt with the Korteweg-de Vries equation, here we apply the proposed spectral method for calculating the soliton solution of (1.7). Some preliminary results in this direction have been obtained in [27]. The situation here is more complicated for two reasons: a higher-order derivative is present, and the solution is neither an even nor an odd function. As such, the solution is sought in the following truncated series:

$$\varphi(x) = \sum_{n=0}^{N} [h_n(x)S_n(x) + \bar{h}_n(x)C_n(x)]. \tag{5.1}$$

Introducing the latter into (1.7), after standard manipulations, we arrive at the nonlinear system for coefficients

$$\begin{aligned} &\sum_{m=0}^{N} (\theta_{nm} + \varphi_{nm}) - ch_n + 6 \sum_{m_1=0}^{N} \sum_{m_2=0}^{N} \gamma_{m_1m_2n} h_{m_1} \bar{h}_{m_2} = 0, \\ &-\sum_{m=0}^{N} (\theta_{nm} + \varphi_{nm}) - c\bar{h}_n + 3 \sum_{m_1=0}^{N} \sum_{m_2=0}^{N} (\alpha_{m_1m_2n} h_{m_1} h_{m_2} + \beta_{m_1m_2n} \bar{h}_{m_1} \bar{h}_{m_2}) = 0. \end{aligned} \tag{5.2}$$

System (5.2) contains $2N+2$ equations for the $2N+2$ unknown coefficients $h_0, \bar{h}_0, \cdots, h_N, \bar{h}_N$. The general scheme of the algorithm is the same as in § 3. We stress that in this case the convergence with respect to the number N is slower and can be seen in Table 3. If we consider only the first seven pairs h_i, $\bar{h}_i$, the convergence within three percent ($\varepsilon = 0.03$) is attained for $N = 20$. The slow convergence is easily explained by the fact that the solution has a more complicated form (see Fig. 5), showing a row of local minima and maxima. It turns out that in the last case the adequate choice of scaling factor β is much more important than in the previous case, where the solution has monotone shape. Here we do not employ the full-scale technique for defining the optimal β. Rather, we estimate from obvious considerations that $\beta = 2$ is to be good enough for "compressing" the soliton to fit the length scale of the S_0, C_0. Indeed, Table 4 and Fig. 6 convince us that the results are considerably improved. First, the accuracy reached with $N = 20$ is $\varepsilon = 0.001$ (30 times better) and this time even $N = 0$ gives fully acceptable approximation for the soliton. For $N = 3$ the obtained accuracy corresponds to $N = 6$ with $\beta = 1$, and $N = 14$ matches the performance of $N = 20$. We should mention that the results above are obtained for $c = 1$.

TABLE 3
Developing the solution with N for $c = 1$.

	i \ $N+1$	7	8	9	11	15	20
	0	0.1936	0.0325	−0.1474	0.0787	0.0297	0.0813
	1	0.0572	0.1104	0.1459	0.0707	0.0969	0.0764
	2	−0.0531	0.0101	0.0880	−0.0114	0.0066	−0.0140
	3	−0.0857	−0.0669	−0.0229	−0.0554	−0.0602	−0.0625
	4	−0.0684	−0.0859	−0.0880	−0.0580	−0.0739	−0.0625
	5	−0.0361	−0.0662	−0.0970	−0.0401	−0.0519	−0.0370
	6	−0.0108	−0.0349	−0.0717	−0.0197	−0.0176	−0.0074
h_i	7		−0.0105	−0.0373	−0.0054	0.0127	0.0145
	8			−0.0112	0.0011	0.0314	0.0250
	9				0.0022	0.0377	0.0256
	10				0.0010	0.0344	0.0199
	11					0.0258	0.0117
	12					0.0159	0.0038
	13					0.0075	−0.0021
	14					0.0021	−0.0053
	0	0.6634	0.6859	0.6267	0.6493	0.6713	0.6632
	1	0.0723	0.1036	0.1407	0.0562	0.0901	0.0710
	2	−0.0745	−0.0726	−0.0616	−0.0913	−0.0786	−0.0832
	3	−0.0713	−0.0887	−0.1027	−0.0708	−0.0774	−0.0706
	4	−0.0381	−0.0563	−0.0776	−0.0141	−0.0306	−0.0199
	5	−0.0135	−0.0247	−0.0409	0.0279	0.0097	0.0188
	6	−0.0026	−0.0069	−0.0148	0.0443	0.0297	0.0344
$\bar{h}_i$	7		−0.0008	−0.0026	0.0409	0.0322	0.0315
	8			0.0003	0.0279	0.0249	0.0186
	9				0.0138	0.0150	0.0032
	10				0.0041	0.0065	−0.0038
	11					0.0013	−0.0182
	12					0.0159	0.0038
	13					−0.0010	−0.0212
	14					−0.0004	−0.0181
		Maximal absolute error compared to the case $N+1=20$					
		0.1714	0.1168	0.1821	0.0212	0.0091	0

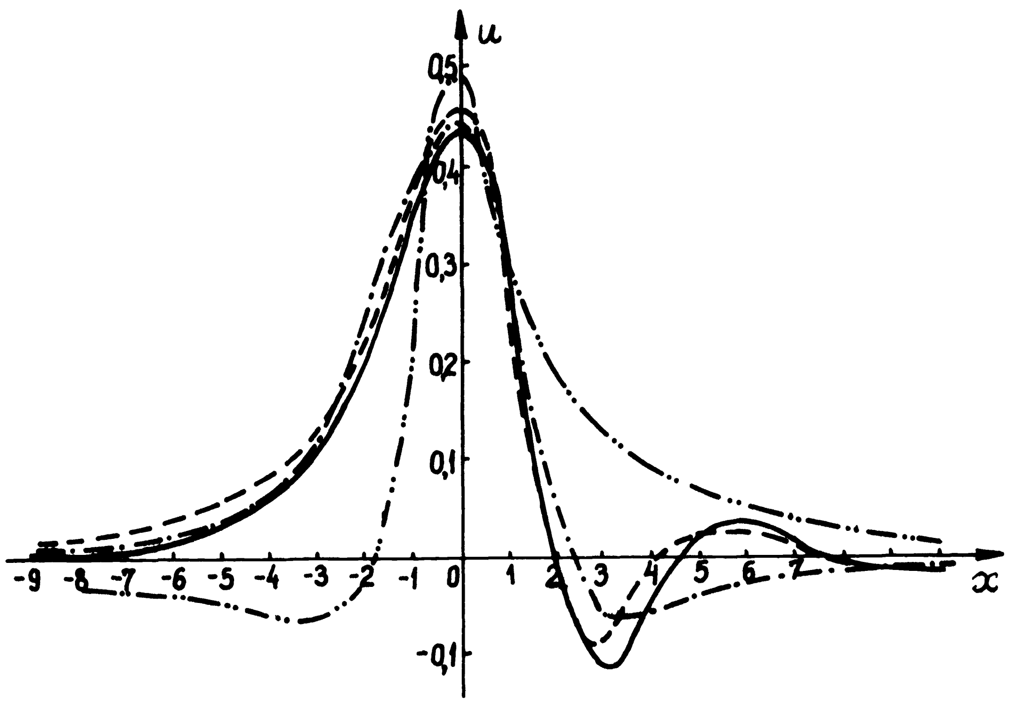


FIG. 5. *Developing the calculated shape of the film soliton for* $c = 1$ *with number of modes* $N+1$: – · · – ($N = 0$); – · – ($N = 6$); – – – ($N = 19$) *and comparison with the difference solution of* [4] ——.

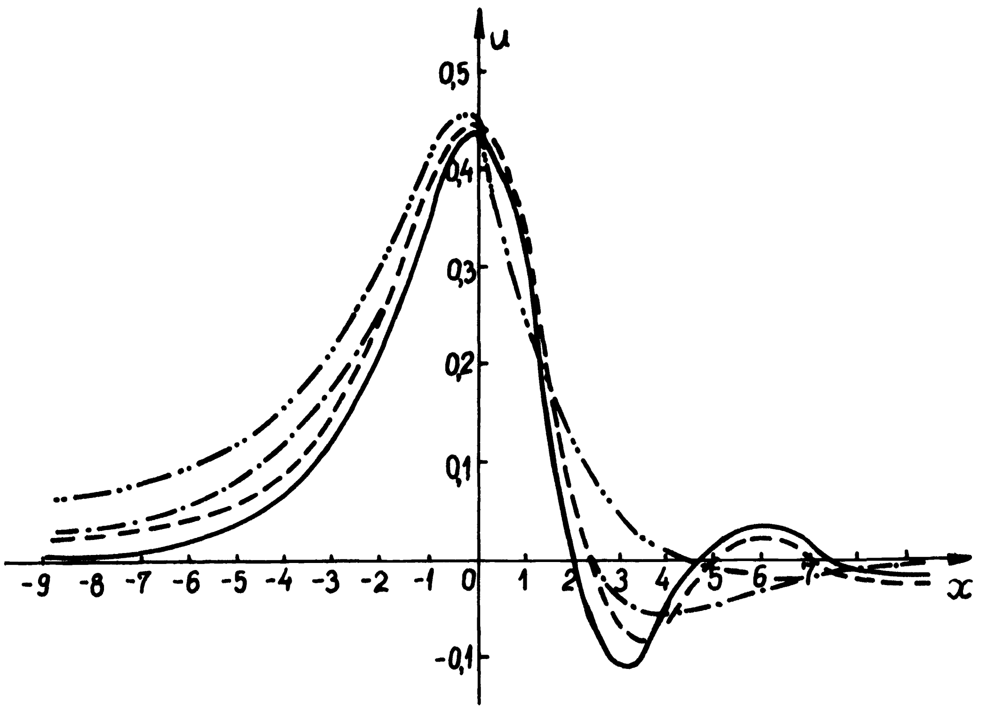


FIG. 6. *Developing the calculated shape of the film soliton for* $c = 1$ *and the scaling factor* $\beta = 2$ *with the number* $N+1$ *of modes*: – · · – ($N = 0$); – · – ($N = 3$); – – – ($N = 14$) *and comparison with the difference solution of* [4] ——.

Now, after being convinced that the proposed method is highly effective for calculating the shapes of solitons, we try to answer the question of what for spectrum exists for the parameter c. In [4] it is found that the difference solution to the problem based on the notion of variational imbedding exists for $0.4 \leqq c \leqq 4.9$ and the results strongly suggest that the soliton is to be expected for the entire open interval $c \in (0, \infty)$. It is not computed in [4] only because some measures for the grid are to be taken for $c \ll 1$ and $c \gg 1$. The present results confirm that conclusion and the shape of the film soliton is obtained for a wide range of governing parameters c (see Fig. 7). We can see the intricate shape for very high $c \geqq 10$. The latter can be thought of, however, only as preliminary results as far as the physics of the phenomenon is concerned, but is still a very good achievement of the method of Fourier in $L^2(-\infty, \infty)$.

6. Conclusions. The present paper deals with developing the numerical aspects of a new technique for the Fourier method in $L^2(-\infty, \infty)$ with a CON basis system of functions proposed earlier. The necessary formulae are compiled and the systems for coefficients of the series are obtained in the frame of the Galerkin approach for two famous nonlinear equations: Korteweg-de Vries and Kuramoto-Sivashinsky for which

TABLE 4
Developing the solution with N for $c = 1$ and the scale factor $\beta = 2$.

	i \ $N+1$	2	4	7	12	15	20
	0	−0.0180	0.2083	0.2351	0.2162	0.2128	0.2112
	1	0.1452	−0.0656	−0.1109	−0.0939	−0.0939	−0.0916
	2		−0.0923	−0.0703	−0.0691	−0.0664	−0.0672
	3		−0.0317	0.0106	0.0062	0.0677	0.0065
	4			0.0357	0.0311	0.0299	0.0297
	5			0.0246	0.0195	0.0167	0.0174
	6			0.0080	0.0009	−0.0012	−0.0002
h_i	7				−0.0101	−0.0098	−0.0091
	8				−0.0120	−0.0089	−0.0089
	9				−0.0087	−0.0036	−0.0041
	10				−0.0044	0.0014	0.0009
	11				−0.0012	0.0004	0.0004
	12					0.0040	0.0042
	13					0.0025	0.0031
	0	0.4378	0.4922	0.4632	0.4698	0.4674	0.4689
	1	−0.1032	−0.0911	−0.0846	−0.0930	−0.0955	−0.0957
	2		−0.0159	−0.0125	−0.0116	−0.0118	−0.0115
	3		0.0129	0.0152	0.0202	0.0223	0.0224
	4			0.0027	0.0025	0.0046	0.0047
	5			−0.0057	−0.0125	−0.0122	−0.0126
	6			−0.0037	−0.0120	−0.0136	−0.0135
$\bar{h}_i$	7				−0.0039	−0.0058	−0.0053
	8				0.0031	0.0024	0.0033
	9				0.0055	0.0071	0.0075
	10				0.0040	0.0078	0.0072
	11				0.0015	0.0061	0.0041
	12					0.0036	0.0004
	13					0.0016	−0.0025
		Maximal absolute error compared to the case $N+1=20$					
		0.2291	0.0280	0.0211	0.0180	0.0011	0

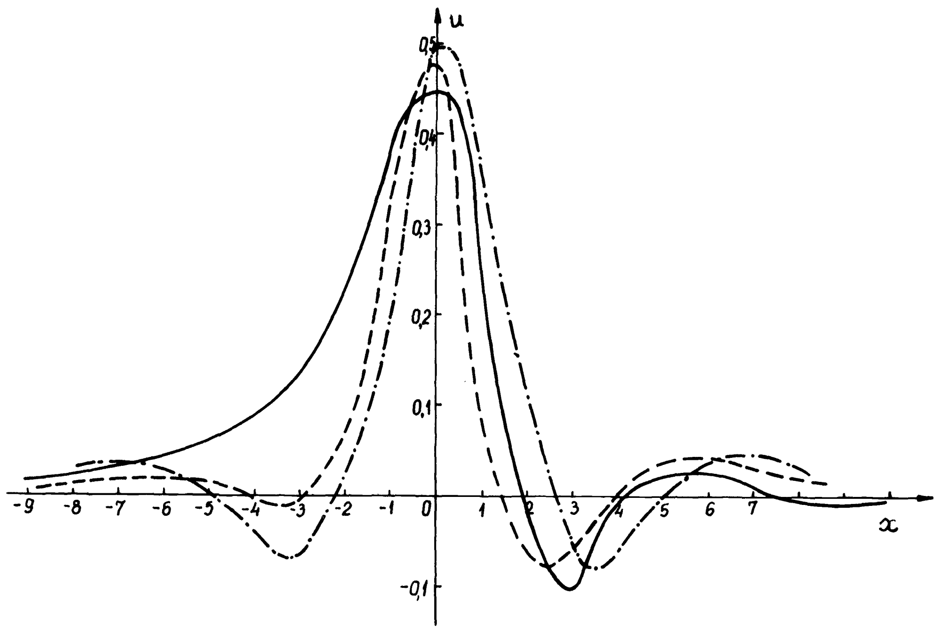


FIG. 7. *Evolution of the soliton shape with the parameter c*: —— ($c=1$); --- ($c=10$); -·- ($c=20$).

soliton solutions do exist. The performance of the Fourier-Galerkin method is checked through comparing the approximate solution for KdV to the known analytic solution. The different kinds of errors (truncation, discretization, and absolute one) are calculated as functions of the spatial coordinate x. Some means for optimization of the Fourier method based on the notion of scaling the independent variable are discussed. Then the mathematical technology is applied to the K-S equation and the soliton solution is obtained for a variety of values of nonlinear eigenvalue parameters c. The shapes of solitons compare well with known difference solutions.

The results obtained suggest that a reliable and robust numerical technique is devised for calculating the shape of solitons occurring as solutions for certain nonlinear differential equations of evolution.

Acknowledgments. The authors are indebted to the referees for helpful suggestions.

REFERENCES

[1] M. A. ABLOWITZ AND H. SEGUR, *Solitons and the Inverse Scattering Transform*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1981.

[2] P. L. BHATNAGAR, *Nonlinear Waves in One-Dimensional Dispersive Systems*, Clarendon Press, Oxford 1979.

[3] C. I. CHRISTOV, *A method for identification of homoclinic trajectories*, in Proc. 14th Spring Conference Union Bulgarian Mathematicians, Sunny Beach, Bulgaria, 1985, pp. 571-577.

[4] C. I. CHRISTOV AND V. P. NARTOV, *The use of a method of variational imbedding for calculating solitons in falling-liquid-film flow*, in Proc. Conference "Anniversary of academician L. Tchakalov," Samokov, Bulgaria, 1986, pp. 135-142. (In Russian.)

[5] H. BATEMAN, *Higher Transcendental Functions*, Vol. 2, McGraw-Hill, New York, Toronto, London, 1953.

[6] C. E. GROSCH AND S. A. ORSZAG, *Numerical solution of problems in unbounded regions: Coordinate transformations*, J. Comput. Phys., 25 (1977), pp. 273-296.

[7] J. P. BOYD, *The optimization of convergence for Chebyshev polynomial methods in an unbounded domain*, J. Comput. Phys., 45 (1982), pp. 43-79.

[8] ———, *Spectral methods using rational basis functions on an infinite interval*, J. Comput. Phys., 69 (1987), pp. 112-142.

[9] C. I. CHRISTOV, *A complete orthonormal system of functions in $L^2(-\infty, \infty)$ space*, SIAM. J. Appl. Math., 42 (1982), pp. 1337-1344.

[10] ———, *A method for treating the stochastic bifurcation of plane Poiseuille flow*, Ann. Univ. Sofia, Fac. Math. Mech., Libre—Mechanique, vol. 76/1982 (1987), pp. 87-113.

[11] S. V. ALEKSEENKO, V. E. NAKORYAKOV, AND B. G. POKUSAEV, *Wave formation on vertical falling liquid films*, Internat. J. Multiphase Flow, 11 (1985), pp. 607-627.

[12] A. PUMIR, P. MANNEVILLE, AND Y. POMEAU, *On the solitary waves running down an inclined plane*, J. Fluid Mech., 135 (1983), pp. 27-50.

[13] D. J. BENNEY, *Long waves in liquid films*, J. Math. Phys., 45 (1966), pp. 150-155.

[14] S. P. LIN, *Finite amplitude side-band stability of a viscous film*, J. Fluid Mech., 63 (1974), pp. 417-429.

[15] G. SIVASHINSKY AND D. M. MICHELSON, *On irregular wavy flow of a liquid down a vertical plane*, Progr. Theoret. Phys., 63 (1980), pp. 2112-2114.

[16] Y. KURAMOTO, *Diffusion-induced chemical turbulence*, in Dyn. Synerg. Syst., Proc. Internat. Symposium Synerg., Biefield, Berlin, 1979, pp. 134-146.

[17] M. HYMAN AND B. NIKOLAENKO, *The Kuramotò-Sivashinsky equation: A bridge between* PDE'*s and dynamical systems*, Phys. D, 18 (1986), pp. 113-126.

[18] V. YA. SHKADOV, *Solitary waves in viscous films*, Izv. Akad. Nauk SSSR, Mekhanika Zhidkosti i Gasa, 1 (1977), pp. 63-66. (In Russian.)

[19] O. YU. TZVELODUB, *Stationary propagating waves on films falling down an inclined plane*, Izv. Akad. Nauk SSSR, Mekh. Zhidk. Gaza, 4 (1980), pp. 142-146. (In Russian.)

[20] N. WIENER, *Extrapolation, Interpolation and Smoothing of Stationary Time Series*, Technology Press MIT and John Wiley, New York, 1949.

[21] J. R. HIGGINS, *Completeness and Basis Properties of Sets of Special Functions*, Cambridge University Press, London, New York, Melbourne, 1977.

[22] P. GESHEV, Private communication, Institute of Theoretical and Applied Mechanics, Siberian Division of Academy of Sciences of the USSR, Novisibirsk, 1986.

[23] J. P. BOYD, *The orthogonal rational functions of Higgins and Christov and algebraically mapped Chebyshev polynomials*, J. Approx. Theory, (1989), in press.

[24] C. CANUTO, M. Y. HUSSAINI, A. QUARTERONI, AND T. A. ZANG, *Spectral Methods in Fluid Dynamics*, Springer-Verlag, New York, Berlin, Heidelberg, London, Paris, Tokyo, 1988.

[25] J. J. MORÉ AND M. Y. COSNARD, BRENTM, *A Fortran Subroutine for the Numerical Solution of a System of Nonlinear Equations*, Collected Algorithms from ACM, Algorithm No. 554, Association for Computing Machinery.

[26] D. J. WILD, *Optimum Seeking Methods*, Prentice-Hall, Englewood Cliffs, NJ, 1964.

[27] C. I. CHRISTOV AND K. L. BEKYAROV, *A Fourier-series method for calculating solitons*, University Annual, Applied Mathematics, Sofia, Bulgaria, 22 (1986), pp. 199-209. (In Russian.)

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 4, pp. 648-655, July 1990

FAST POLAR DECOMPOSITION OF AN ARBITRARY MATRIX*

NICHOLAS J. HIGHAM† AND ROBERT S. SCHREIBER‡

Abstract. The polar decomposition of an $m \times n$ matrix A of full rank, where $m \geqq n$, can be computed using a quadratically convergent algorithm of Higham [*SIAM J. Sci. Statist. Comput.*, 7 (1986), pp. 1160-1174]. The algorithm is based on a Newton iteration involving a matrix inverse. It is shown how, with the use of a preliminary complete orthogonal decomposition, the algorithm can be extended to arbitrary A. The use of the algorithm to compute the positive semidefinite square root of a Hermitian positive semidefinite matrix is also described. A hybrid algorithm that adaptively switches from the matrix inversion based iteration to a matrix multiplication based iteration due to Kovarik, and to Björck and Bowie, is formulated. The decision when to switch is made using a condition estimator. This "matrix multiplication rich" algorithm is shown to be more efficient on machines for which matrix multiplication can be executed 1.5 times faster than matrix inversion.

Key words. polar decomposition, complete orthogonal decomposition, matrix square root, matrix multiplication, Schulz iteration, condition estimator

AMS(MOS) subject classification. 65F05

C.R. classification. G.1.3

1. Introduction. A polar decomposition of a matrix $A \in \mathbf{C}^{m\times n}$ is a factorization $A = UH$, where $H \in \mathbf{C}^{n\times n}$ is Hermitian positive semidefinite and $U \in \mathbf{C}^{m\times n}$ is unitary; here we define unitary to mean that U has orthonormal rows or columns according as $m \leqq n$ or $m \geqq n$. The decomposition always exists, H is the unique Hermitian positive semidefinite square root of A^*A (i.e., $H = (A^*A)^{1/2}$), and U is unique if and only if A has full rank (these properties are proved in § 2).

The polar decomposition is well known in the case $m \geqq n$ (see [8] and [11], for example). We have followed Horn and Johnson [14] in extending the definition to $m \leqq n$. The consistency of the definition can be seen in the result that for any m and n the unitary polar factor U is a nearest unitary matrix to A in the Frobenius norm (this is a straightforward extension of a result from [6]). Because of the role it plays in solving this and other nearness problems, computation of the polar decomposition is required in several applications [13]. A recent application, which motivated the work here, is the computation of block reflectors (generalizations of Householder matrices) [19]. Here, the polar decomposition of an arbitrary matrix must be computed, and it is desirable to do this efficiently on vector and parallel computers.

The polar decomposition can be obtained directly from the singular value decomposition (SVD). Higham [11] describes an alternative approach based on a Newton iteration involving a matrix inverse. The iteration is defined for square, nonsingular matrices only, but in [11] it is pointed out how a preliminary QR decomposition enables the treatment of $A \in \mathbf{C}^{m\times n}$ with $m \geqq n$ and rank $(A) = n$. It is also shown in [11] how the iteration can be used to compute the square root of a Hermitian positive definite matrix. According to the traditional model of computational cost based on operation counts, the iterative algorithm is generally of similar expense to

* Received by the editors November 14, 1988; accepted for publication (in revised form) July 12, 1989.

† Department of Computer Science, Upson Hall, Cornell University, Ithaca, New York 14853. This research was conducted while the author was on leave from the Department of Mathematics, University of Manchester, Manchester M13 9PL, United Kingdom.

‡ Research Institute for Advanced Computer Science (RIACS), National Aeronautics and Space Administration (NASA) Ames Research Center, Moffett Field, California 94035. The research of this author was partially supported by Office of Naval Research contract N00014-86-K-0610.

the SVD approach, but is much more efficient when the matrix is nearly unitary. In an attempt to improve the performance of the iterative algorithm on machines that execute matrix multiplication at high efficiency, Schreiber and Parlett [19] propose the use of an inner Schulz iteration to compute most of the matrix inverses; they show that this leads to an increase in efficiency if matrix multiplication can be done at a rate 6.8 times faster than matrix inversion. Gander [7] develops a family of iteration methods for computing the polar decomposition of a rectangular matrix of full rank; his family includes a variant of the Newton iteration of [11].

The purpose of this work is twofold. First, we extend the algorithm of [11] so that it is applicable to arbitrary A. Our technique is to use an initial complete orthogonal decomposition so as to extract an appropriate square, nonsingular matrix. We might say that the complete orthogonal decomposition is to the polar decomposition what Chan's preliminary QR factorization is to the SVD! We also show how to use the algorithm of [11] to compute the square root of a (singular) Hermitian positive semidefinite matrix. Second, we introduce a modification of the Schulz inner iteration idea of [19] that reduces the cutoff ratio of multiplication speed to inversion speed from 6.8 to 2, or to 1.5 if advantage is taken of a symmetric matrix product.

2. Iterative polar decomposition of an arbitrary matrix. The basic algorithm of [11] is as follows. It converges quadratically for any square, nonsingular A. We use a MATLAB-like algorithmic notation, and denote by A^{-*} the conjugate transpose of A^{-1}.

```
ALGORITHM 2.1. $[U, H] = \text{polar} \cdot \text{square}(A, \delta)$.
% Input arguments: square, nonsingular $A$; convergence tolerance $\delta$.
% Output arguments: $U$, $H$.
$X_0 = A$; $k = -1$
repeat
    $k = k + 1$
    $\gamma_k = (\|X_k^{-1}\|_1 \|X_k^{-1}\|_\infty / (\|X_k\|_1 \|X_k\|_\infty))^{1/4}$
    $X_{k+1} = \frac{1}{2}(\gamma_k X_k + X_k^{-*}/\gamma_k)$
until $\|X_{k+1} - X_k\|_1 \leq \delta \|X_{k+1}\|_1$
$U = X_{k+1}$
$H = \frac{1}{2}(U^* A + A^* U)$
```

To adapt the algorithm to arbitrary $A \in \mathbf{C}^{m \times n}$ we begin by computing a complete orthogonal deomposition (COD)

$$A = P \begin{bmatrix} R & 0 \\ 0 & 0 \end{bmatrix} Q^*,$$

where $P \in \mathbf{C}^{m \times m}$ and $Q \in \mathbf{C}^{n \times n}$ are unitary, and $R \in \mathbf{C}^{r \times r}$ is nonsingular and upper triangular (we exclude the trivial case $A = 0$, for which R is empty). This decomposition may be computed using a QR factorization with column pivoting followed by a further Householder reduction step (see [9, p. 169] for the details). Now we apply Algorithm 2.1 to R, obtaining $R = U_R H_R$, and we "piece together" the polar factors of A. We have

$$\begin{aligned} A &= P \begin{bmatrix} U_R H_R & 0 \\ 0 & 0 \end{bmatrix} Q^* \\ &= P \begin{bmatrix} U_R & 0 \\ 0 & I_{m-r,n-r} \end{bmatrix} \begin{bmatrix} H_R & 0 \\ 0 & 0 \end{bmatrix} Q^* \\ &= P \begin{bmatrix} U_R & 0 \\ 0 & I_{m-r,n-r} \end{bmatrix} Q^* \cdot Q \begin{bmatrix} H_R & 0 \\ 0 & 0 \end{bmatrix} Q^* \\ &\equiv UH, \end{aligned}$$

where $I_{m-r,n-r}$ denotes the $(m-r)\times(n-r)$ identity matrix. Note that $I_{m-r,n-r}$ could be replaced by any unitary matrix of the same dimensions; this shows the nonuniqueness of the unitary polar factor when $r=\text{rank}(A)<\min(m,n)$. Note also that even though $U^*U\neq I$ when $m<n$, $H=U^*UH$ for all m and n; thus $A^*A=HU^*UH=H^2$, so that $H=(A^*A)^{1/2}$.

In evaluating U and H advantage can be taken of the zero blocks in the products. Denoting by Q_1 the first r columns of Q, we have

$$H=Q_1H_RQ_1^*. \tag{2.1}$$

For U we partition

$$P=(\overset{r}{P_1},\ \overset{m-r}{P_2})$$

and we distinguish the two cases:

$$m\geqq n \;\Rightarrow\; U=[P_1,P_2]\begin{bmatrix} U_R & 0\\ 0 & \begin{bmatrix} I_{n-r}\\ 0\end{bmatrix}\end{bmatrix}Q^*=\left[P_1U_R,\ \ P_2\begin{bmatrix} I_{n-r}\\ 0\end{bmatrix}\right]Q^*, \tag{2.2a}$$

$$m\leqq n \;\Rightarrow\; U=[P_1,P_2]\begin{bmatrix} U_R & 0\\ 0 & [I_{m-r},0]\end{bmatrix}Q^*=[P_1U_R,\ \ P_2[I_{m-r},0]]Q^*, \tag{2.2b}$$

in which, respectively, the last $m-n$ columns of P_2 and the last $n-m$ rows of Q^* need not participate in the multiplication.

To summarise, we have the following algorithm.

```
ALGORITHM 2.2. [$U$, $H$] = polar ($A$, $\varepsilon$, $\delta$).
% $A\neq 0$ is arbitrary.
[$P$, $R$, $Q$] = COD ($A$, $\varepsilon$)
[$U_R$, $H_R$] = polar · square ($R$, $\delta$)
Form $U$, $H$ according to (2.1) and (2.2).
```

As the notation indicates, in floating-point arithmetic a tolerance ε is required for the complete orthogonal decomposition to determine a numerical rank (i.e., the dimension of R). The natural approach is to set to zero all rows of the trapezoidal QR factor of A that are negligible (in some measure) relative to ε (see [9, p. 166]). The choice of ε is important, since a small change in ε can produce a large change in the computed U when A is rank-deficient. However, a redeeming feature is that whatever the choice of ε, and irrespective of how well the QR factorization reveals rank, Algorithm 2.2 is stable, that is, the computed polar factors $\hat{U}$, $\hat{H}$ satisfy

$$\hat{U}\hat{H}=A+E,$$

where $\|E\|_F\approx\max\{\varepsilon,\delta\}\|A\|_F$; this follows from the empirical stability of Algorithm 2.1 (see [11]) together with the stability of the additional orthogonal transformations in Algorithm 2.2.

The operation count of Algorithm 2.2 breaks down as follows, using the "flop" notation [9, p. 32]. The complete orthogonal decomposition requires $2mnr-r^2(m+n)+2r^3/3+r^2(n-r)$ flops [9, pp. 165, 170]. Algorithm 2.1 requires, typically, eight iterations (assuming $\delta\approx 10^{-16}$), and hence $(7+\frac{2}{3})r^3$ flops (taking into account the triangularity of R). And formation of H and U requires at most $nr^2+n^2r/2$ and $mr^2+\max\{nm^2, mn^2\}$ flops, respectively. By comparison, computing a polar decomposition via the Golub-Reinsch SVD algorithm requires approximately $8mn^2+25n^3/6$ flops when $m \geqq n$. The Golub-Reinsch SVD algorithm does not take advantage of rank-deficiency, although it could be modified to do so by using an initial complete orthogonal decomposition as above.

Of course, operation counts are not always a reliable guide to the actual computational cost on modern vector and parallel computers. An alternative performance indicator is the amount of matrix multiplication in an algorithm, since matrix multiplication can be performed very efficiently on many modern computers [1], [3], [18], [20]. As we will see in the next section, Algorithm 2.1 can be modified so that it is rich in matrix multiplication. In the complete orthogonal decomposition in Algorithm 2.2 the second Householder reduction step can be accomplished using the matrix multiplication rich WY representation of [3], [20]. In the initial QR factorization effective use of the WY representation is precluded by the column pivoting. One alternative is to use Bischof's local pivoting and incremental condition estimation technique [2], which does not hinder exploitation of the WY form. Another alternative is to compute a QR factorization *without* pivoting, and then to apply Chan's post-processing algorithm [5] for obtaining a rank-revealing QR factorization.

Finally, we show how to use Algorithm 2.1 to compute the Hermitian positive semidefinite square root of a Hermitian positive semidefinite $A \in \mathbf{C}^{n\times n}$. First, we compute a Cholesky decomposition with pivoting,

$$\Pi^T A \Pi = R^* R, \qquad R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix},$$

where $R_{11} \in \mathbf{C}^{r\times r}$ is nonsingular and upper triangular. Then Householder transformations are used to zero R_{12} (as in the complete orthogonal decomposition):

$$U^*\Pi^T A \Pi U = \begin{bmatrix} T_{11}^* \\ 0 \end{bmatrix} [T_{11} \quad 0], \qquad T_{11} \in \mathbf{C}^{r\times r} \text{ upper triangular.}$$

Next, Algorithm 2.1 applied to T_{11} yields $T_{11} = U_T H_T$, whence, with $Q = \Pi U$,

$$A = Q \begin{bmatrix} H_T^2 & 0 \\ 0 & 0 \end{bmatrix} Q^* = \left(Q \begin{bmatrix} H_T & 0 \\ 0 & 0 \end{bmatrix} Q^* \right)^2 \equiv X^2.$$

Square roots of semidefinite matrices are required in some statistical applications [10]. An alternative to this polar decomposition approach is to make use of an eigendecomposition; the relative merits are similar to those discussed above for the SVD.

3. A hybrid iteration. To make Algorithm 2.1 rich in matrix multiplication rather than matrix inversion, Schreiber and Parlett [19] use an inner Schulz iteration,

$$(3.1) \qquad Z_{j+1} = Z_j + (I - Z_j X_k) Z_j, \qquad Z_0 = X_{k-1}^{-1},$$

to compute X_k^{-1} on all iterations after the first. This approach takes advantage of the fact that since the X_k are converging quadratically, X_{k-1}^{-1} is an increasingly good approximation to X_k^{-1}. The Schulz iteration (3.1) is a Newton iteration and so also converges quadratically. Schreiber and Parlett observe that for the matrices in their application (which are often well conditioned) the typical number of inner iterations required for convergence is 6, 5, 3, 2, 1, leading to 17 iterations in total, or 34 matrix multiplications. If the matrix inverses were computed directly, five inverses would be needed. This suggests that the modified algorithm will be faster than Algorithm 2.1 if matrix multiplication can be done at a rate 34/5 times faster than matrix inversion.

Further experimentation with the inner Schulz iteration led us to feel that it is unnecessary to run the inner iteration to convergence, and we considered employing just *one* Schulz iteration, with the starting matrix $Z_0 = X_k^*$ ($\approx X_k^{-1}$ since X_k converges to a unitary matrix). Thus the basic iteration

$$X_{k+1} = \frac{1}{2}\left(\gamma_k X_k + \frac{1}{\gamma_k} X_k^{-*}\right) \tag{3.2}$$

is replaced by (setting $\gamma_k = 1$)

$$\begin{aligned} X_{k+1} &= \tfrac{1}{2}(X_k + (Z_0^* + Z_0^*(I - Z_0 X_k)^*)) \\ &= X_k(I + \tfrac{1}{2}(I - X_k^* X_k)). \end{aligned} \tag{3.3}$$

This is precisely the quadratically convergent iteration of Kovarik [15] and Björck and Bowie [4] for computing the unitary polar factor! Hence, just a single inner Schulz iteration is enough to maintain quadratic convergence.

The convergence of (3.3) is described by the following relation, noted in [17]:

$$R_{k+1} = \tfrac{3}{4}R_k^2 + \tfrac{1}{4}R_k^3,$$

where $R_k = I - X_k^* X_k$. (Using this relation, we can show that the asymptotic error constant is $\frac{3}{2}$ for (3.3) compared with $\frac{1}{2}$ for (3.2) [11].) If $\|R_k\| < 1$, then

$$\|R_{k+1}\| < \tfrac{3}{4}\|R_k\|^2 + \tfrac{1}{4}\|R_k\|^2 = \|R_k\|^2 < \|R_k\|.$$

To maximise the number of matrix multiplications we therefore need to switch from iteration (3.2) to iteration (3.3) as soon as the convergence condition

$$\|X_k^* X_k - I\| \leqq \theta < 1 \tag{3.4}$$

is satisfied; to ensure fast convergence θ should not be too close to 1. As explained below, typically (3.4) is satisfied for $k = 3$ with $\theta = 0.6$ (and obviously for $k = 0$ if $X_0 = A$ happens to be nearly unitary). Rather than expend a matrix multiplication testing (3.4) we can use the matrix norm estimator CONEST from [12]. This computes a lower bound for $\|C\|_1$ by sampling several matrix-vector products Cx and C^*x; thus we can estimate $\|X_k^* X_k - I\|_1$, without forming $X_k^* X_k$, in $O(r^2)$ flops (for $r \times r$ X_k). A suitable way to use the estimate is to test whether it is less than $\lambda\theta$, where $\lambda < 1$. If so, $X_k^* X_k - I$ is formed, in preparation for (3.3), and its norm is taken. If (3.4) is satisfied then (3.3) is used—otherwise we revert to iteration (3.2). The optimum choice of λ depends on the desired bias between wasting a matrix multiplication in an abortive switch of iteration, and not switching soon enough. The estimate from CONEST is almost always correct to within a factor 3, so $\lambda \geqq \frac{1}{3}$ is appropriate. In practice we have found that the performance of the algorithm is fairly insensitive to the choices of θ and λ.

To summarise, our hybrid inversion/multiplication algorithm is as follows.

```
ALGORITHM 3.1. $[U, H] = \text{polar} \cdot \text{mult}\,(A, \delta, \lambda, \theta)$.
% $A$ must be a square, nonsingular matrix.
$X_0 = A$; $k = -1$; $\mu = 1$; switched = false
repeat
```

```
        $k = k+1$
        if switched
            $R = I - X_k^* X_k$; $\mu = \|R\|_1$
            evaluate (3.3)
        else
            $\mu = \mathrm{CONEST}\,(I - X_k^* X_k)$
            if $\mu > \lambda\theta$
                evaluate (3.2)
            else
                $R = I - X_k^* X_k$; $\mu = \|R\|_1$
                if $\mu > \theta$, evaluate (3.2), else evaluate (3.3), switched = true; end
            end
        end
    until $\mu \leqq \delta$
    $U = X_{k+1}$
    $H = \frac{1}{2}(U^* A + A^* U)$
```

Since iteration (3.3) requires two matrix multiplications, and iteration (3.2) requires one inversion, Algorithm 3.1 will be more efficient than Algorithm 2.1 if matrix multiplication can be done at twice the rate of matrix inversion; thus, compared with using the full inner Schulz iteration, the "cutoff ratio" is 2 instead of 6.8. Moreover, if advantage is taken of the symmetry of the second matrix product in (3.3) the cutoff ratio is reduced to 1.5. The overall speedup depends on the ratio of inversions to multiplications, which in turn depends on the conditioning of the matrix, as discussed below.

All the algorithms mentioned here have been coded and tested in PC-MATLAB [16], running on an IBM PC-AT. For this machine the unit roundoff $u \approx 2.22 \times 10^{-16}$. We used $\theta = .6$, $\lambda = .75$, $\delta = \sqrt{r}\, u$, where r is the dimension of the matrix A in Algorithms 2.1 and 3.1, and $\varepsilon = \max(m, n)|t_{11}|u$ in the complete orthogonal decomposition, where T is the triangular factor from the QR factorization with complete pivoting.

The following comments summarise our numerical experience, based on a wide variety of test matrices.

- Algorithms 2.1 and 3.1 usually require the same number of iterations. Occasionally Algorithm 3.1 requires one more iteration due to the larger error constant for iteration (3.3).
- In general, the typical number of iterations for Algorithm 3.1 is seven to nine, within the switch of iteration on iteration three or four.

TABLE 3.1

k	$\kappa_F(X_{k+1})$	$\\|X_k^* X_k - I\\|_1$ [1]	Iteration	γ_k
0	2.6265E7	1.1380E10	(3.2)	3.1546E−3
1	5.3197E2	2.6233E4	(3.2)	8.0931E−3
2	4.0886E0	8.0962E−2 [2]	(3.3)	
3	3.9959E0	4.4915E−3	(3.3)	
4	4.0000E0	1.3686E−5	(3.3)	
5	4.0000E0	1.2607E−10	(3.3)	
6	4.0000E0	1.5765E−17	(3.3)	

[1] Norm estimate when (3.2) is used; exact quantity when (3.3) is used.
[2] Norm estimate exact to five digits.

• For well-conditioned matrices ($\kappa_2(A) \leqq 10$, say), as are common in certain applications (see [13]), Algorithm 3.1 tends to require at most seven iterations and the switch is on iteration one, two, or three.

We present the results for one representative matrix in detail: MATLAB's "gallery(5)," which is the 5×5 nilpotent matrix

$$A=\begin{bmatrix} -9 & 11 & -21 & 63 & -252 \\ 70 & -69 & 141 & -421 & 1684 \\ -575 & 575 & -1149 & 3451 & -13801 \\ 3891 & -3891 & 7782 & -23345 & 93365 \\ 1024 & -1024 & 2048 & -6144 & 24572 \end{bmatrix}.$$

Using Algorithm 3.1 within Algorithm 2.2, the numerical rank is diagnosed as 4, and Algorithm 3.1 is presented with a triangular matrix having one singular value of order 10^5 and three of order 1. Table 3.1 summarises the iteration. The backward error $\|A-\hat{U}\hat{H}\|_1 = 4.7u\|A\|_1$.

Acknowledgment. We thank Des Higham for suggesting several improvements to the manuscript.

REFERENCES

[1] D. H. BAILEY, *Extra high speed matrix multiplication on the* Cray-2, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 603-607.

[2] C. H. BISCHOF, QR *factorization algorithms for coarse-grained distributed systems*, Ph.D. thesis, Cornell University, Ithaca, NY, 1988.

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

[4] Å. BJÖRCK AND C. BOWIE, *An iterative algorithm for computing the best estimate of an orthogonal matrix*, SIAM J. Numer. Anal., 8 (1971), pp. 358-364.

[5] T. F. CHAN, *Rank revealing* QR *factorizations*, Linear Algebra Appl., 88/89 (1987), pp. 67-82.

[6] K. FAN AND A. J. HOFFMAN, *Some metric inequalities in the space of matrices*, Proc. Amer. Math. Soc., 6 (1955), pp. 111-116.

[7] W. GANDER, *Algorithms for the polar decomposition*, Manuscript, Institut fuer Informatik, ETH, Zürich, 1988.

[8] F. R. GANTMACHER, *The Theory of Matrices*, Vol. 1, Chelsea, New York, 1959.

[9] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, 1983.

[10] J. C. GOWER, *Multivariate analysis: Ordination, multidimensional scaling and allied topics*, in Handbook of Applicable Mathematics, Vol. VI: Statistics, E. H. Lloyd, ed., John Wiley, Chichester, 1984, pp. 727-781.

[11] N. J. HIGHAM, *Computing the polar decomposition—with applications*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1160-1174.

[12] ———, FORTRAN *codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation*, ACM Trans. Math. Software, 14 (1988), pp. 381-396.

[13] ———, *Matrix nearness problems and applications*, in Applications of Matrix Theory, M. J. C. Gover and S. Barnett, eds., Oxford University Press, Oxford, 1989, pp. 1-27.

[14] R. A. HORN AND C. R. JOHNSON, *Matrix Analysis*, Cambridge University Press, Cambridge, 1985.

[15] Z. KOVARIK, *Some iterative methods for improving orthonormality*, SIAM J. Numer. Anal., 7 (1970), pp. 386-389.

[16] C. B. MOLER, J. N. LITTLE, AND S. BANGERT, PC-MATLAB *User's Guide*, The MathWorks, Inc., 21 Eliot St., South Natick, MA, 01760, 1987.

[17] B. PHILIPPE, *An algorithm to improve nearly orthonormal sets of vectors on a vector processor*, SIAM J. Algebraic Discrete Methods, 8 (1987), pp. 396-403.

[18] R. S. SCHREIBER, *Block algorithms for parallel machines*, in Numerical Algorithms for Modern Parallel Computer Architectures, M. H. Schultz, ed., IMA Volumes In Mathematics and Its Applications 13, Springer-Verlag, Berlin, 1988, pp. 197-207.

[19] R. S. SCHREIBER AND B. N. PARLETT, *Block reflectors: Theory and computation*, SIAM J. Numer. Anal., 25 (1988), pp. 189-205.

[20] R. S. SCHREIBER AND C. F. VAN LOAN, *A storage efficient* WY *representation for products of Householder transformations*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 53-57.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 4, pp. 656-687, July 1990

ON SHAPE-PRESERVING INTERPOLATION AND SEMI-LAGRANGIAN TRANSPORT*

PHILIP J. RASCH† AND DAVID L. WILLIAMSON†

Abstract. A large number of interpolation schemes are evaluated in terms of their relative accuracy. The large number of schemes arises by considering combinations of interpolating forms (piecewise cubic polynomials, piecewise rational quadratic and cubic polynomials, and piecewise quadratic Bernstein polynomials), derivative estimates (Akima, Hyman, arithmetic, geometric and harmonic means, and Fritsch-Butland), and modification of these estimates required to ensure monotonicity and/or convexity upon the interpolant. *Shape-preserving* methods maintain in the interpolant the monotonicity and/or convexity implied in the discrete data.

The schemes are first compared by evaluating their ability to interpolate evenly spaced data drawn from three test shapes (Gaussian, cosine bell, and triangle) at two resolutions. Details of the cosine bell tests are presented in this paper. Details of the other tests are presented in a companion technical report. Of the monotonic interpolants, the following are the most accurate: (1) The Hermite cubic interpolant with the derivative estimate of Hyman modified to produce monotonicity as suggested by de Boor and Swartz. (2) The second version of the rational cubic spline suggested by Delbourgo and Gregory, with the derivative estimate of Hyman modified to produce monotonicity. (3) The piecewise quadratic Bernstein polynomials suggested by McAllistor and Roulier with the derivative estimate of Hyman again modified. Imposing strict monotonicity at discrete extrema introduces significant errors. More accurate interpolations result if this requirement is relaxed at extrema. The Hermite cubic interpolant is improved by relaxing the strict monotonicity constraint to one suggested by Hyman at extrema. In a like manner, the accuracy of the rational and piecewise quadratic Bernstein polynomial interpolants can be improved by requiring only that convexity/concavity be satisfied rather than monotonicity.

Some of the more accurate interpolants are incorporated into the semi-Lagrangian transport method and tested by examining the accuracy of the solution to one-dimensional advection of test shapes in a uniform velocity field. The semi-Lagrangian method using monotonic interpolators provides monotonic solutions. The semi-Lagrangian method using interpolators that maintain convex/concave constraints give solutions that are *essentially nonoscillatory.* The monotonic forms damp the solution with time, more so for narrow than broad structures. The best monotonic forms are the Hermite cubic interpolant with the Akima or Hyman derivative estimates modified to produce monotonicity with C^0 continuity. The corresponding C^1 continuous forms have unacceptable phase errors with the Hermite interpolant. The rational cubic with the Hyman derivative estimate modified to produce monotonicity is comparable to the C^0 Hermite form described above. The C^1 rational form does not have the phase error seen in the C^1 Hermite interpolant. The essentially nonoscillatory forms damp much less than the monotonic forms. The solutions that used rational cubic interpolants with a Hyman derivative estimate modified to satisfy a convexity/concavity constraint were the most satisfactory of the shape-preserving schemes.

Key words. monotone, convex, interpolation, shape preserving, advection, transport

AMS(MOS) subject classifications. 65D05, 65D07, 65M05

1. Introduction.

Shape-preserving interpolation denotes a class of methods that maintain any monotonicity, and/or convexity suggested by data in the interpolant. These shape-preserving properties provide a means of avoiding the oscillations often seen in polynomial interpolation. Many methods [4], [15], [16], [17], [9], [5], [6], [12], [8], [7] have been introduced in the past few years with shape-preserving properties. They have usually been evaluated in terms of their "visually pleasing" nature, or via the error terms of the associated Taylor series. While these quantities

* Received by the editors March 7, 1988; accepted for publication (in revised form) July 25, 1989. This work was partially supported by U.S. Navy grant N66685686WR86062.

† National Center for Atmospheric Research, Climate and Global Dynamics Division, P.O. Box 3000, Boulder, Colorado 80307-3000. The National Center for Atmospheric Research is sponsored by the National Science Foundation.

are of great importance, many problems require the interpolation of data that are *not* strictly monotonic, or convex. Often the underlying form of the data has discontinuities in its derivatives, and the Taylor series error estimates are of limited utility. Thus, it is desirable to evaluate the accuracy of the interpolant in a more general context. In this paper we objectively compare shape-preserving interpolants with each other and with non-shape-preserving forms to provide a sound basis for choosing one for a particular application.

This paper attempts to provide a survey of a number of the currently accepted methods of shape-preserving interpolation that have appeared in the literature. We present a review of the methods, and then use two classes of tests to allow the schemes to be compared. Because of the review of methods, a careful description of the experimental design, and discussion of the results, the paper is long. Some of the supporting details are provided in a companion technical report [21]. We have made a strong effort to condense our results and conclusions into a concise form in § 5, which we urge the reader to peruse first. That section contains no "numbers." Results are reviewed, and schemes are ordered qualitatively and quantitatively. We are not aware of any study that compares such a broad variety of schemes using objective tests. This was the primary motivation for performing the study. We needed to know of a reasonably accurate, simple, shape preserving interpolation scheme.

The ultimate application requiring this study was to use a shape-preserving interpolant as one part of a set of improvements to a technique known in the meteorological literature as semi-Lagrangian transport (cf., [19], the references therein, and the references of § 4). Semi-Lagrangian transport is used for (among other things) the solution of differential equations describing the evolution of scalar fields embedded in an advecting velocity field.

It is difficult to construct semi-Lagrangian schemes that strictly conserve mass. To balance this disadvantage they have the desirable property of unconditional computational stability, resulting in schemes that do not have the very common restriction, often referred to as the Courant number restriction, on the timestep of $U\delta t/\delta x \leqq 1$, where U is the velocity, and δt and δx are intervals of time and space discretization. They also are not generally cursed with the slowdown in phase speed with large Courant number often seen in implicit forms of computational methods whose explicit cousins have a Courant number restriction. The semi-Lagrangian schemes have been shown in [18] to be consistent, and this property, with the stability property mentioned earlier implies convergence via the Lax Equivalence Theorem. We are constrained only by accuracy considerations in the choice of timestep length, with no additional stability considerations. We were led to a consideration of semi-Lagrangian methods in a set of calculations on the sphere with the computational grid constrained by other aspects of the problem [30] to be nearly equally spaced in latitude and longitude. With this computational grid we invariably encounter flows in which the Courant number greatly exceeds 1, and we needed a computational method that would deal with this situation.

One part of the semi-Lagrangian scheme requires an accurate interpolation of the scalar field, and it was this interpolation that we hoped to improve using a shape-preserving technique. By replacing the polynomial interpolants usually used in semi-Lagrangian transport with shape-preserving interpolants, the physically unreasonable oscillations seen in solutions to this equation should be reduced or eliminated. With this improvement, the semi-Lagrangian scheme becomes closer to the monotonic [2], [26], [27], [28], [29], or total variation diminishing schemes [10], or positive-definite transport algorithms [24]. These schemes are often used for solving evolution equations posed in flux form. Although the shape-preserving forms of semi-Lagrangian transport schemes and monotonic or total variation diminishing schemes mentioned above do not show the undesirable oscillations—in exchange they are characterized by increased damping, diffusion, or clipping phenomena over the oscillatory solutions. While any numerical scheme can be made to provide positive-definite solutions by ad hoc corrections to the solution at the end of each timestep, these corrections cannot be applied to the corresponding overshooting in the solution. We have found that in practice their application to schemes that are oscillatory often leads to substantial errors in the solution as well. The transport scheme must obviously be chosen to suit the problem at hand, and often it is more important to maintain the positivity, or nonoscillatory nature of the solution, and sacrifice other characteristics. In this paper we explore the trade-offs seen in moving from nonshape-preserving forms of semi-Lagrangian transport to shape-preserving forms.

In § 2 a notation is established, and the various shape-preserving interpolation schemes are reviewed. Various interpolation schemes are compared in § 3, in terms of their accuracy in representing a single test shape. A more complete test with shapes that differ in their degree of continuity, monotonicity, and convexity is provided in [21]. A summary of the interpolation comparison of § 3 and [21] appears in § 3.4. The schemes judged better in this comparison are then used within the semi-Lagrangian technique (§ 4) in the solution to the simplest of transport problems, that of one-dimensional advection of a scalar field (of prescribed initial shape) by a constant velocity field. In our tests involving this equation, the error is associated only with the interpolation. The accuracy of the solution thus provides an objective measure of the error accumulated over many interpolations. The overall conclusions are provided in § 5. These tests allowed us to compare many of the shape-preserving interpolation forms, and we judged the results to be of probable interest to the community using these interpolation formulae and to the community interested in improvements to semi-Lagrangian transport.

2. The interpolation problem. We begin by defining the grid $\{x_i\}_{i=1}^{n}$, $x_1 < x_2 < \cdots < x_n$, and the data values $\{f_i\}, f_i = f(x_i)$. It is also convenient to define the *discrete slopes*

$$\Delta_i = \frac{f_{i+1} - f_i}{x_{i+1} - x_i}. \tag{1}$$

The data are locally monotonic on the double grid interval $[x_{i-1}, x_{i+1}]$ if

$$\Delta_{i-1}\Delta_i > 0, \tag{2}$$

and locally convex if

$$\Delta_{i-1} > \Delta_i. \tag{3}$$

For concave data, the previous inequality (3) is reversed. We note that with these definitions, some data may be interpreted as concave and convex on a single grid interval. We deal with this special case in § 3.2. We define the piecewise interpolant $p \in \mathscr{C}^{\mathscr{K}}[x_1, x_n]$, with $K \geqq 0$. On each subinterval $[x_i, x_{i+1}]$, let

$$\theta = \frac{x - x_i}{h_i}, \qquad h_i = x_{i+1} - x_i, \tag{4}$$

and

$$p(x) = p_i(\theta). \tag{5}$$

The interpolant p has the following properties:

$$p(x_i) = f_i, \qquad \frac{dp(x_i)}{dx} = d_i. \tag{6}$$

Here, d_i is some estimate of the derivative of f at the endpoints of the subinterval. The interpolant is specified on the subinterval in terms of the data f_i, and the derivative estimates d_i at the endpoints of the subinterval, that is,

$$p_i(\theta) = p_i(\theta, f_i, f_{i+1}, d_i, d_{i+1}). \tag{7}$$

The interpolant thus adheres to the standard osculatory representation, although the functional form of p is not necessarily the usual Hermite form of the cubic polynomial. For the intercomparison, only interpolating forms that involve use of *local* information are included, i.e., d_i is a function of a few surrounding values of f_i. In this fashion we have excluded from consideration many global schemes; for example, the classic C^2 cubic splines which minimize the integral of the curvature of the interpolant over the entire domain, exponential splines under tension [25], and global versions of the monotone, piecewise interpolants of [8] and [5]. These schemes require information over the entire domain. We chose to evaluate local methods because our major final goal was a local transport scheme. Local methods are also desirable because adding, changing, or removing data in the domain will only change the interpolant in the vicinity of the change of data. Following this restriction, schemes that differ from each other in three major ways are considered:

- The method of estimating the derivative is varied according to algorithms that have appeared in the shape-preserving literature.
- The type of interpolating function is varied to encompass cubic polynomials, rational functions, and quadratic Bernstein polynomials with extra knots.
- To guarantee monotonicity or concavity/convexity in the interpolating function, certain constraints are imposed on the derivative estimates. The appropriate constraint depends on the interpolation form.

It is convenient to address these items in reverse order in the following sections.

2.1. Constraints on the derivatives. Certain constraints must be imposed on the derivative estimates used in the interpolation schemes for the interpolants to maintain the properties of convexity/concavity or monotonicity present in the data. The constraints are reviewed in this section, proceeding from the least to the most restrictive form. The constraints can be written in terms of restrictions on the derivative estimates d at the endpoints of an interval, as a function of the discrete slope Δ within the interval. Because of this, the constraint on d_i based on Δ_{i-1} of the interval to the left may be different from that based on Δ_i of the interval to the right. We may choose to use a different derivative estimate at a point for interpolation over two adjacent intervals by constraining the estimate differently, in which case the interpolant is C^0, or insist that constraints associated with both intervals be satisfied simultaneously, in which case the same derivative estimate is used for the adjacent intervals, and the interpolant is C^1. When the constraint on d_i depends not only on the discrete slopes over the adjacent intervals Δ_{i-1} and Δ_i, but also the derivative estimate d_{i-1} or d_{i+1} at the other ends of the intervals, the C^1 interpolants become global. Such forms are not considered in this report.

The requirement that the continuous derivative estimates bound the discrete slope for a C^0 interpolant

$$(d_i - \Delta_i)(\Delta_i - d_{i+1}) > 0 \qquad \text{(NCC0)} \tag{8}$$

and lie between the adjacent discrete slopes for a C^1 interpolant

$$(d_i - \Delta_{i-1})(\Delta_i - d_i) > 0 \qquad \text{(NCC1)} \tag{9}$$

must be true if the interpolant is to be convex/concave in the intervals $[x_i, x_{i+1}]$ and $[x_{i-1}, x_{i+1}]$, respectively. These requirements are identified as *Necessary Condition(s) for Convexity/Concavity*, C^0 and C^1, respectively.

In order that the interpolating function be monotonic on the interval $[x_i, x_{i+1}]$ and C^0, the derivatives must satisfy the *Necessary Condition for Monotonicity* C^0

$$\begin{array}{ll} \operatorname{sign}(d_i) = \operatorname{sign}(\Delta_i) = \operatorname{sign}(d_{i+1}), & \Delta_i \neq 0, \\ d_i = d_{i+1} = 0, & \Delta_i = 0, \end{array} \qquad \text{(NCM0)} \tag{10}$$

that is, the derivative estimate at the endpoints must have the same sign as the discrete slope on the interval. For a C^1 interpolant on the interval $[x_{i-1}, x_{i+1}]$

$$\begin{array}{ll} \operatorname{sign}(\Delta_{i-1}) = \operatorname{sign}(d_i) = \operatorname{sign}(\Delta_i), & \Delta_{i-1}\Delta_i > 0, \\ d_i = 0, & \Delta_{i-1}\Delta_i \leqq 0, \end{array} \qquad \text{(NCM1).} \tag{11}$$

The derivative estimate at the data point must have the same sign as the discrete slopes surrounding it or be zero if the discrete datum is an extremum at this point. This condition is the *Necessary Condition for Monotonicity* C^1 (NCM1).

For the rational and piecewise quadratic Bernstein polynomial interpolation forms discussed below, the necessary conditions NCM0 and NCM1 are also sufficient conditions for monotonicity. Similarly, the NCC0 and NCC1 are sufficient conditions for convexity with these interpolants. On the other hand, for Hermite cubic interpolants NCM0 and NCM1 are necessary but not sufficient for monotonicity and must be augmented by additional constraints on the derivatives.

Fritsch and Carlson [8] have found both necessary and sufficient conditions for monotonicity of Hermite cubic interpolants. Let $\alpha = d_i/\Delta_i$, $\beta = d_{i+1}/\Delta_i$; then if $\Delta \neq 0$ the cubic interpolant will be monotonic if and only if (α, β) lies within the domain $\mathcal{M}_{ns}$ defined by the union of two domains:

$$\mathcal{M}_{ns} = \mathcal{M}_e \cup \mathcal{M}_b \tag{12}$$

where

$$\mathcal{M}_e(\alpha, \beta) = \{\alpha, \beta \colon \phi(\alpha, \beta) \leqq 0\}, \tag{13}$$

$$\mathcal{M}_b(\alpha, \beta) = \{\alpha, \beta \colon 0 \leqq \alpha \leqq 3, 0 \leqq \beta \leqq 3\}, \tag{14}$$

and

$$\phi(\alpha, \beta) = (\alpha - 1)^2 + (\alpha - 1)(\beta - 1) + (\beta - 1)^2 - 3(\alpha + \beta - 2). \tag{15}$$

If $\Delta_i = 0$, then $d_i = d_{i+1} = 0$ and the necessary condition discussed earlier is also sufficient. Embedded in this domain $\mathcal{M}_{ns}$ is the region $\mathcal{M}_b$ recognized independently by de Boor and Swartz [4] that provides a sufficient condition for monotonicity for the Hermite cubic interpolant. This sufficient condition

$$0 \leqq \alpha \leqq 3, \qquad 0 \leqq \beta \leqq 3 \qquad \text{(SCM)} \tag{16}$$

is easier to apply than the more general necessary and sufficient condition ($\mathcal{M}_{ns}$) in which α and β may be dependent on each other. Throughout the remainder of this article, this simpler condition will be referred to as the *Sufficient Condition for Monotonicity* (SCM). As before, we define C^0 and C^1 forms depending on whether the derivatives d_i are bounded by just the Δ of the interval being interpolated or by the Δ of the two adjacent intervals simultaneously.

Constraints on the derivative estimates are applied in the following fashion. The NCM constraints are imposed according to

$$d_i \leftarrow \begin{cases} d_i, & d_i\Delta_j > 0, \\ 0, & d_i\Delta_j \leqq 0 \end{cases} \tag{17}$$

where $i=j, j+1$ for NCM0 interpolation on the interval $[x_j, x_{j+1}]$, and $j=i-1, i$ for NCM1 interpolation on the double interval $[x_{i-1}, x_{i+1}]$. Similarly, the SCM constraints use

$$d_i \leftarrow \text{SIGN}\,(d_i) \min\,(|d_i|, |3\Delta_j|) \tag{18}$$

where $i=j, j+1$ for SCM0, and $j=i-1, i$ for SCM1, and are applied following the corresponding NCM constraint. Finally, we use the following algorithm to apply the NCC1 constraint.

$$d_i \leftarrow \begin{cases} d_i, & (d_i-\Delta_{i-1})(\Delta_i-d_i) \geqq 0, \\ d_{lim}, & (d_i-\Delta_{i-1})(\Delta_i-d_i) < 0 \end{cases} \tag{19}$$

where

$$d_{lim} = \begin{cases} \min\,(\Delta_{i-1}, \Delta_i), & d_i < \Delta_{i-1}, \\ \max\,(\Delta_{i-1}, \Delta_i), & d_i \geqq \Delta_{i-1}. \end{cases} \tag{20}$$

At an extremum where the data are not monotonic over the surrounding double interval, NCM1 limiting provides a severe restriction, as d_i is constrained to be zero there. The interpolant must put the extremum at the data point. For Hermite cubic interpolants Hyman [12] has relaxed the SCM1 limiting concept where the data reach a local extremum, and are not monotonic, in an attempt to mimic a convexity constraint. He proposed the following limit on the derivatives:

$$d_i \leftarrow \text{SIGN}\,(d_i) \min\,(|d_i|, |3\Delta_{i-1}|, |3\Delta_i|). \tag{21}$$

This allows for overshoot on the interval next to local discrete extrema and thus is nonmonotonic, but does provide some control of the overshoot and, in particular, prevents oscillations at the edge of flat plateaus. This type of limit is more in the spirit of essentially nonoscillatory schemes as introduced by Harten and Osher [10] in which overshooting can occur to prevent undesirable clipping, but no additional extrema are allowed.

2.2. Interpolation forms. Three types of interpolating functions are considered, all of which appeared in the recent literature regarding shape preserving interpolation:

- Cubic polynomials [4], [8], [12], [7]
- Quadratic Bernstein polynomials with extra knots [15], [16], [17]
- Rational functions [9], [5], [6].

The Hermite cubic and rational interpolating functions can be described using the formalism of Delbourgo and Gregory [5]. Consider the function

$$p_i = \frac{P_i(\theta)}{Q_i(\theta)} \tag{22}$$

on the interval $0 \leqq \theta \leqq 1$, equivalently $x_i \leqq x \leqq x_{i+1}$, where

$$P_i(\theta) = f_{i+1}\theta^3 + (r_i f_{i+1} - h_i d_{i+1})\theta^2(1-\theta) + (r_i f_i + h_i d_i)\theta(1-\theta)^2 + f_i(1-\theta)^3 \tag{23}$$

and

$$Q_i(\theta)=1+(r_i-3)\theta(1-\theta). \tag{24}$$

We consider four choices of the parameter r_i:

- If $r_i=3$, p_i reduces to the standard *Hermite cubic* polynomial interpolation form. Recall that the interpolant will be monotonic if the d_i lie within the domain M_b.
- If $r_i=1+(d_i+d_{i+1})/\Delta_i$, then P_i and Q_i reduce to quadratic polynomials, and p_i is identified as a rational quadratic interpolant. Delbourgo and Gregory [5] have shown that provided d_i and d_{i+1} satisfy the NCM0, p_i will be monotonic over the subinterval, otherwise this interpolant is not well defined. If NCM0 (10) is not satisfied we modify the derivatives to satisfy it in order to apply the rational quadratic via (17).
- If $r_i=1+\max(C_i/c_i, C_i/c_{i+1})$, where $c_i=\Delta_i-d_i$, $c_{i+1}=d_{i+1}-\Delta_i$, $C_i=d_{i+1}-d_i$ then P_i is a cubic polynomial, and p_i is identified as the rational cubic interpolant version 1.
- If $r_i=1+c_{i+1}/c_i+c_i/c_{i+1}$, then P_i is again a cubic polynomial and p_i is identified as the rational cubic interpolant version 2. Delbourgo and Gregory [5] have shown that if the derivatives satisfy the convexity/concavity constraints NCC0 or NCC1 then both rational cubic versions will be convex/concave. If, in addition, the derivatives

TABLE 1
Algorithms for derivative estimates as they simplify for evenly spaced data.

Identifier	Algorithm
Akima [1], [8], [12]	$d_i=\begin{cases}\dfrac{\alpha\Delta_{i-1}+\beta\Delta_i}{\alpha+\beta}, & \alpha+\beta\neq 0\\ \dfrac{(\Delta_{i-1}+\Delta_i)}{2}, & \alpha+\beta=0\end{cases}$ $\alpha=\lvert\Delta_{i+1}-\Delta_i\rvert,\ \beta=\lvert\Delta_{i-1}-\Delta_{i-2}\rvert$
Arithmetic mean [8], [9], [5], [12] Deficient spline	$d_i=\dfrac{(\Delta_{i-1}+\Delta_i)}{2}$
Geometric mean [5]	$d_i=\begin{cases}\operatorname{sign}(\Delta_i)\sqrt{\Delta_{i-1}\Delta_i}, & \Delta_{i-1}\Delta_i\geqq 0\\ 0, & \Delta_{i-1}\Delta_i<0\end{cases}$
Harmonic mean [7] Rational linear [9] McAllister-Roulier [17]	$d_i=\begin{cases}\dfrac{2\Delta_{i-1}\Delta_i}{(\Delta_{i-1}+\Delta_i)}, & \Delta_{i-1}\Delta_i\geqq 0\\ 0, & \Delta_{i-1}\Delta_i<0\end{cases}$
Fritsch-Butland [7], [12]	$d_i=\begin{cases}\dfrac{3\lvert\Delta_{i-1}\rvert\lvert\Delta_i\rvert}{\max(\Delta_{i-1},\Delta_i)+2\min(\Delta_{i-1},\Delta_i)}, & \Delta_{i-1}\Delta_i\geqq 0\\ 0, & \Delta_{i-1}\Delta_i<0\end{cases}$
Cubic	$d_i=\begin{cases}\dfrac{(2\Delta_{i-1}+5\Delta_i-\Delta_{i+1})}{6}, & x\in(x_i,x_{i+1})\\ \dfrac{(-\Delta_{i-2}+5\Delta_{i-1}+2\Delta_i)}{6}, & x\in(x_{i-1},x_i)\end{cases}$
Hyman [12]	$d_i=\dfrac{\Delta_{i-2}-7\Delta_{i-1}+7\Delta_i-\Delta_{i+1}}{12}$

satisfy the monotonicity constraints NCM0 or NCM1, then both versions will be monotonic. Delbourgo and Gregory [5] have also shown that version 2 is, in general, more accurate than version 1. The derivative estimates must satisfy NCC0 for the two versions of the rational cubic interpolant to be well defined.

The quadratic Bernstein polynomials with extra knots cannot be described using the previous formalism. This interpolant is constructed by piecing together two quadratic Bernstein polynomials within each interval, with the point of intersection (the extra knot) determined by a rather complex algorithm that cannot be succinctly described with a few equations or figures. Because of this, the reader should refer to the descriptions found in the series of original articles [15], [16], [17]. The characteristics of the Bernstein polynomials, together with the algorithms developed for constructing the knot, the value of the interpolant at the knot, and the interpolant derivative at the knot guarantee that the interpolant will be monotonic provided NCM is satisfied, and convex/concave provided NCC is satisfied.

2.3. Derivative estimation procedures. Table 1 lists the algorithms used in estimating derivatives at the nodes. Several of the algorithms suggested in the literature for shape preserving interpolation that differ for unequally spaced data reduce to a common form when the data become equally spaced. The tests that follow use only equally spaced data, and therefore common algorithms are grouped together. The table also includes an algorithm identified as Cubic, which does not usually appear as a derivative estimate. This scheme arises by computing a cubic interpolant through the four points surrounding the interval. The derivative estimates at the ends of the interval can then be written as a linear combination of the four surrounding data points. Such a scheme results in an interpolant that is only C^0 continuous. It is included because this form of interpolation is often used in semi-Lagrangian problems. The harmonic mean, geometric mean, and Fritsch-Butland derivative estimates automatically satisfy the NCM and NCC constraints. The others generally must be modified to satisfy them as described in § 2.1.

3. Intercomparisons of interpolation schemes.

3.1. Test shapes and diagnostics. The accuracy of the interpolation schemes has been tested on a uniform grid using three shapes: Gaussian, cosine bell, and triangle. For the cosine and triangle shapes, the nonzero portion is confined to the central half of the domain. These shapes were chosen because they have similar forms, but may be successively more difficult to approximate accurately. The Gaussian is C^∞, the cosine bell C^1, and the triangle C^0. The tests were made by embedding the shapes within the domain $[0, 2\pi]$. Tests using 10 and 40 intervals over the 2π domain were performed. The shapes were successively displaced 100 times, by 1/100 of the grid interval and measurements of the accuracy were made over the domain $[0, 2\pi]$. This was to establish the sensitivity of the representation to the relative position of the grid and test shape. The error varied by at least a factor of 5 over the 100 realizations. We compare the schemes using the error averaged over all realizations. For brevity this paper documents the results only for the well-resolved cosine bell shape. The other cases are documented in [21].

More precisely, define the data points in the evaluation domain $[0, 2\pi]$ to be $x_l = (l-1)h$, where $h = 2\pi/N$, $l = 1, N+1$, and $N = 10$ or 40 for the two widths chosen. The test functions are given at these data points and the interpolation is evaluated over the set of points within $[0, 2\pi]$ given by $\tilde{x}_j = (j-1)h/13$, $j = 1, 13N+1$. Note, $x_l = \tilde{x}_{13(l-1)+1}$ and the data points where the interpolators fit exactly are included in the error measures. Extra data points were added outside the domain, when needed to compute the appropriate derivative estimates near zero and 2π. The exact forms for the functions to be interpolated were

Gaussian:

$$F(x)=\exp\left[-2\left(\frac{(x-c_n)}{\delta}\right)^2\right]; \tag{25}$$

Cosine Bell:

$$F(x)=\begin{cases}\frac{1}{2}\left\{1+\cos\left(\frac{\pi(x-c_n)}{\delta}\right)\right\}, & |x-c_n|<\delta,\\ 0, & |x-c_n|\geqq\delta;\end{cases} \tag{26}$$

Triangle:

$$F(x)=\begin{cases}\left(1-\frac{|x-c_n|}{\delta}\right), & |x-c_n|<\delta,\\ 0, & |x-c_n|\geqq\delta\end{cases} \tag{27}$$

where $\delta=10\pi/24$ specifies the width of the test shapes and $c_n=\pi-(n-1)h/100+\varepsilon$ controls the offset of the test shape with respect to the grid. The additional small offset $\varepsilon=10^{-5}$ is included so that the maximum of the test functions never coincides with a sampled point. The error over the entire domain $[0,2\pi]$, denoted the *total error*, is given by

$$E_T=\frac{1}{100}\sum_{n=1}^{100}\sum_{j=1}^{13N+1}[P(\tilde{x}_j)-F(\tilde{x}_j,c_n)]^2h/13. \tag{28}$$

We also consider the error over the domain $[0,2\pi]$ excluding the two grid intervals adjacent to the discrete maximum of the test shape. The test shapes are monotonic over this evaluation domain and we refer to this error as the *monotonic region error*

$$E_M=\frac{1}{100}\sum_{n=1}^{100}\sum_{j\in\mathcal{M}}[P(\tilde{x}_j)-F(\tilde{x}_j,c_n)]^2h/13 \tag{29}$$

where

$$\mathcal{M}=[j;j<j_L,j>j_R] \tag{30}$$

and the left and right bounds of the excluded gridpoints are determined from the grid intervals where the discrete slope of the test function changes sign, i.e., given the data point l^* at which

$$\Delta_{l^*-1}\Delta_{l^*}<0, \tag{31}$$

the bounds are given by

$$j_L=13(l^*-2)+1, \tag{32}$$

$$j_R=13(l^*)+1. \tag{33}$$

3.2. Description of interpolation schemes and notation. The error statistics are presented in the tables as a function of interpolation form, derivative approximation scheme, and derivative limiter. The least restrictive derivative limiter leading to a well-posed interpolant is always included, even if it allows over/undershooting. Limiters are then included in the table that are expected to result in improvements in the interpolants, by reduction of the over/undershooting. The various forms of the interpolation are reviewed in the next few paragraphs and a naming convention for use in the tables is introduced.

We have included seven types of derivative estimates in the intercomparison: the Akima estimate (AKI—[1], [8], [12]), Arithmetic mean (ARI—[8], [9], [5], [12]), Geometric mean (GEO—[5]), Harmonic mean estimate (HAR—[7], [9], [17]), Fritsch-Butland estimate (BUT—[7], [12]), the derivative estimate that gives rise to the simple piecewise cubic interpolant achieved by fitting the cubic through the data nearest the point of current interest (CUB), and the Hyman estimate (HYM—[12]).

The first spline form used is the *Hermite cubic interpolant.* Monotonic forms are constructed by applying the sufficient conditions (SCM0 and SCM1). Hyman's extension to the SCM1 limiter, HYM1, which allows limited overshooting in the vicinity of extrema, is included. As mentioned before, relaxation of the monotonicity condition at the discrete extrema is in the spirit of essentially nonoscillatory schemes that allow for an extremum to form between data points, but do not allow additional extrema to form. Hyman's version can be relaxed further by applying no limiting in the vicinity of extrema, that is, the sufficient condition for monotonicity is applied only in the vicinity of monotonic data (SCM0-EE and SCM1-EE). The EE notation is used to imply "Except at Extrema."

As discussed in § 2, the rational interpolant forms require less stringent derivative limiters. The *rational quadratic interpolant* is properly posed only when the derivative estimates satisfy NCM0. Like the Hermite cubic interpolant, the limiters are applied to constrain the derivatives in the whole domain, and except at extrema. For the NCM1-EE case, the NCM0 constraint is applied at the extrema to keep the scheme well defined.

As McAllister and Roulier [16] have pointed out, there are inevitable complications that arise when the data and corresponding derivative estimates switch between monotonic and concave/convex states. This change in the character of the data also requires a switching in the way the interpolants are constructed. McAllister and Roulier [16] and coauthors in [15] and [17] have described a complete implementation for the piecewise Bernstein polynomials. The scheme is well posed for all data, but becomes monotonic, or convex/concave only if the derivative estimate satisfies the necessary conditions for monotonicity NCM or convexity/concavity NCC, respectively. Recall that the *Rational cubic interpolants* (versions 1 and 2) are properly posed only if the derivative estimates satisfy the convexity condition NCC0. If the data do not satisfy this condition (in the vicinity of an inflection point), some switching condition must be used. We have implemented this switching within the rational cubic interpolants in the following way.

- The derivative estimates are made;
- The derivative limiter (if any) is applied;
- The data and derivative estimate are used to see if NCC0 is satisfied, if so, the rational cubic interpolant is used, if not, the rational quadratic interpolant, as described above, is used there.

The tables of the errors provide a staggering amount of information, and the discussion justifying the conclusions about the interpolation from the tables is somewhat tedious. Therefore, most of the discussion and tables are relegated to [21]. In the next section, only the errors for one test shape are discussed, as an example of the method of analysis. The detailed discussion of the remaining shapes appears in [21]. The results are then summarized in this paper for all the tests (including those of [21]) in the following section and the important conclusions about the interpolation intercomparisons are highlighted there.

3.3. Well-resolved cosine test shape.

3.3.1. Hermite cubic (Table 2). The errors associated with the Hermite cubic interpolation of the cosine bell shape over a domain resolved with 40 points appear in Table 2. Each column provides an indication of the error associated with the application of a different limiting form on the derivative. Each row describes the error associated with the use of a different derivative estimation scheme. The unbracketed number is the error calculated over the whole domain, and the number in brackets is the subset of the errors calculated over the monotonic portion of the domain, i.e., eliminating the two grid intervals adjacent to the discrete maximum in the test data.

When no limiters are applied (first column), the Hyman derivative estimation scheme has the least error, followed by the cubic then the arithmetic mean derivative estimates. In this case, the ordering agrees with the ordering by formal accuracy of the derivative estimates. The ordering by accuracy for the rest of the estimates becomes Akima, geometric mean, Fritsch–Butland, and harmonic mean. We have entered the BUT and HAR forms in the SCM1 column since they automatically satisfy that condition. When the error over only the monotonic part of the domain is considered, the geometric derivative estimate improves greatly in the ranking, with similar changes in the other derivative estimate schemes that automatically satisfy the NCM. Comparing the two types of errors (bracketed and unbracketed) for estimates that automatically satisfy NCM with those that do not suggest that more than half of the error is concentrated near the extrema for the derivative estimates satisfying the NCM, with only a few percent of the error concentrated there for the other schemes.

Application of the SCM degrades the approximations over the whole interval (second or third column compared to first), more so for C^1 continuity (third column) than for C^0 (second column). The only exception is the arithmetic derivative estimate

TABLE 2

Error measures for the Hermite interpolant, cosine bell shape, using 40*-point resolution. Unbracketed numbers represent the ensemble average of the error integral associated with the* 100 *realizations of the shape. The numbers within square brackets represent the error excluding the intervals adjacent to the extrema.*

Slope	No limiter	SCM0	SCM1	HYM1	SCM0-EE	SCM1-EE
Ari	1.25 (−6)	2.43 (−6)	2.71 (−6)	1.54 (−6)	1.22 (−6)	9.66 (−7)
	[1.24 (−6)]	[8.85 (−7)]	[6.32 (−7)]	[6.32 (−7)]	[8.85 (−7)]	[6.32 (−7)]
Cub	1.07 (−6)	2.22 (−6)	2.80 (−6)	1.60 (−6)	1.01 (−6)	1.02 (−6)
	[1.06 (−6)]	[6.78 (−7)]	[6.80 (−7)]	[6.80 (−7)]	[6.78 (−7)]	[6.80 (−7)]
Aki	2.55 (−6)	4.04 (−6)	5.27 (−6)	3.93 (−6)	2.83 (−6)	3.25 (−6)
	[2.52 (−6)]	[2.52 (−6)]	[2.94 (−6)]	[2.94 (−6)]	[2.52 (−6)]	[2.94 (−6)]
But			4.01 (−6)			
			[1.46 (−6)]			
Geo	3.22 (−6)	3.38 (−6)	3.44 (−6)			
	[1.05 (−6)]	[1.07 (−6)]	[1.14 (−6)]			
Har			5.70 (−6)			
			[2.65 (−6)]			
Hym	7.05 (−7)	1.96 (−6)	2.75 (−6)	1.52 (−6)	7.50 (−7)	9.17 (−7)
	[7.05 (−7)]	[4.36 (−7)]	[6.03 (−7)]	[6.03 (−7)]	[4.36 (−7)]	[6.03 (−7)]

in the monotonic region. For all but the geometric derivative estimate, this degradation occurs in the vicinity of the extrema, since the error over the interval excluding the extrema actually decreases with the application of the SCM limiters, less so for C^1 continuity than for C^0.

In general, the application of the monotonicity condition to monotonic data improves the interpolation. In the vicinity of the extrema on the other hand, the error increases, because the monotonic derivative estimates must be zero there and the extrema must occur at a data point rather than in the interior of a subinterval. In the monotonic regions C^0 continuity provides a more accurate interpolation than C^1 continuity (second column versus third) at the expense of smoothness of the interpolant. With the application of the SCM limiters, the advantage of the Hyman over the cubic and arithmetic derivative estimates decreases, although the Hyman remains superior. With the SCM1 limiter, the cubic derivative estimate falls behind the arithmetic derivative estimate, i.e., the C^1 condition compared to the C^0 condition does more harm to the cubic than to the arithmetic derivative estimates (third column versus second).

As mentioned earlier, the Fritsch–Butland and harmonic derivative estimates automatically satisfy SCM1 and so are placed in the third column of Table 2 with the schemes to which they are comparable. Neither is as good as any of the better derivative approximations (Hyman, cubic, or arithmetic) in either the monotonic region or the vicinity of the extrema. The geometric approximation to the derivatives is also not as good as these.

The monotonicity condition associated with the SCM1 requires the derivative estimate to be zero at extrema in the discrete data. The condition will naturally introduce errors when the function has a maximum between the discrete data points. This mismatch in structure is reflected in the error table by errors over the whole interval being larger than those over the monotonic region. The last three columns in Table 2 are schemes that allow for overshooting of a nonmonotonic interpolant in one of the two intervals adjacent to the discrete extrema, the particular interval being the one in which the derivative estimates at the end have opposite sign. The amount of overshoot for the schemes labeled EE is not controlled except inherently by the derivative estimates themselves. The derivative limiter suggested by Hyman permits limited overshooting but eliminates it when the data imply an approach to a flat plateau structure.

The relaxation of the strict monotonicity condition at the extrema to allow overshooting improves the interpolant there with no effect in the strictly monotonic region. For example, the SCM0-EE limiter (Table 2, column 5) has less error over the entire interval than the SCM limiter applied over the entire interval (Table 2, column 2). The errors over the monotonic interval remain the same. Thus the interpolant is improved for all derivative estimates by relaxing the monotonicity condition at extrema. Imposition of Hyman's limiter at the extremum does degrade the accuracy of the interpolant slightly (fourth column versus sixth) in the region of the extrema such that it falls between the limited and nonlimited cases.

To summarize, for the 40 point cosine bell with a Hermite cubic interpolant the Hyman derivative approximation always produces the best interpolation, followed in descending order by cubic, and arithmetic derivative estimates. Imposition of the monotonicity condition either implicitly as in the Fritsch–Butland and harmonic mean estimate, or explicitly with the SCM limiters, improves the interpolation of monotonic data, but can degrade the interpolation near the discrete extrema by suppressing overshooting when the underlying field actually overshoots the discrete values. SCM0 does less damage than SCM1. At the risk of over/undershooting, the limiters should not be applied at the extrema. The C^1 Hyman limiter will allow over/undershooting at isolated extrema but will prevent it on approaching flat plateaus.

3.3.2. Rational quadratic (Table 3). As with the Hermite cubic interpolant, the best derivative estimate is the one proposed by Hyman, followed by the cubic derivative estimate and then the arithmetic (except the C^1 monotonic form where arithmetic is insignificantly better than cubic) estimate. The C^1 form of the approximation is better than the C^0 in the monotonic interval (bracketed terms of the second column versus the first), but not as good in the intervals near the extremum. Relaxing the necessary condition improves the interpolant there (third column versus second). The Akima, Fritsch–Butland, geometric, and harmonic derivative estimates are not as good as the others.

Comparing Tables 2 and 3 shows that the rational quadratic is not as accurate as the Hermite cubic, i.e., comparison of columns one and two of Table 3 with columns two and three of Table 2, respectively. The rational quadratic interpolant is not, in general, as accurate as the monotonic forms of the cubic interpolant. The only exception is the Hyman derivative estimate with C^1 continuity in the monotonic region. The interpolant can exceed the accuracy of the Hermite interpolant for monotonic data, with an accurate derivative estimate.

3.3.3. Rational cubic interpolation forms (Tables 4 and 5). Comparison of the two forms of the rational cubic in the tables entry by entry shows that the second version (Table 5) is consistently better than the first (Table 4) except for a few cases involving the Akima or Fritsch–Butland approximations. Since the Akima, Fritsch–Butland, geometric, and harmonic approximations result in larger errors than the other schemes, we do not consider them further in conjunction with the rational cubic. Therefore, we consider only the second form of the rational cubic (Table 5) coupled with the Hyman, cubic, and arithmetic derivative estimates.

For each derivative limiter (i.e., each column) the Hyman derivative approximation continues to provide the best interpolant, followed by the cubic then arithmetic estimate. Again, they are ordered following their formal accuracy. As with the Hermite cubic,

TABLE 3
Error measure for the rational quadratic interpolant, cosine bell shape, using 40-point resolution. See Table 2 caption for an explanation of the numbers.

Slope	NCM0	NCM1	NCM1-EE
Ari	2.89 (−6)	3.35 (−6)	2.28 (−6)
	[1.24 (−6)]	[6.39 (−7)]	[6.39 (−7)]
Cub	2.47 (−6)	3.36 (−6)	2.25 (−6)
	[8.32 (−7)]	[6.10 (−7)]	[6.10 (−7)]
Aki	4.38 (−6)	5.69 (−6)	4.38 (−6)
	[2.78 (−6)]	[2.78 (−6)]	[2.78 (−6)]
But		5.19 (−6)	
		[2.04 (−6)]	
Geo		4.36 (−6)	
		[1.45 (−6)]	
Har		6.80 (−6)	
		[3.52 (−6)]	
Hym	2.28 (−6)	3.16 (−6)	2.02 (−6)
	[6.60 (−7)]	[3.96 (−7)]	[3.96 (−7)]

TABLE 4

Error measure for the rational cubic interpolant version 1, *cosine bell shape, using* 40-*point resolution. See Table* 2 *caption for an explanation of the numbers.*

Slope	No limiter	NCM0	NCM1	NCM0-EE	NCM1-EE	NCC1
Ari	2.37 (−6)	4.00 (−6)	3.08 (−6)	2.37 (−6)	1.33 (−6)	2.37 (−6)
	[2.35 (−6)]	[2.35 (−6)]	[1.31 (−6)]	[2.35 (−6)]	[1.31 (−6)]	[2.35 (−6)]
Cub	1.19 (−6)	2.62 (−6)	2.54 (−6)	9.85 (−7)	7.62 (−7)	1.06 (−6)
	[1.18 (−6)]	[9.72 (−7)]	[7.50 (−7)]	[9.72 (−7)]	[7.50 (−7)]	[1.05 (−6)]
Aki	2.26 (−6)	3.84 (−6)	4.19 (−6)	2.26 (−6)	2.26 (−6)	2.26 (−6)
	[2.24 (−6)]	[2.24 (−6)]	[2.24 (−6)]	[2.24 (−6)]	[2.24 (−6)]	[2.24 (−6)]
But			3.40 (−6)			
			[1.11 (−6)]			
Geo			3.88 (−6)			
			[1.64 (−6)]			
Har			6.61 (−6)			
			[3.17 (−6)]			
Hym	8.50 (−7)	2.40 (−6)	2.25 (−6)	7.78 (−7)	4.55 (−7)	8.49 (−7)
	[8.49 (−7)]	[7.77 (−7)]	[4.54 (−7)]	[7.77 (−7)]	[4.54 (−7)]	[8.48 (−7)]

TABLE 5

Error measure for the rational cubic interpolant version 2, *cosine bell shape, using* 40-*point resolution. See Table* 2 *caption for an explanation of the numbers.*

Slope	No limiter	NCM0	NCM1	NCM0-EE	NCM1-EE	NCC1
Ari	1.73 (−6)	3.20 (−6)	2.56 (−6)	1.73 (−6)	8.27 (−7)	1.73 (−6)
	[1.72 (−6)]	[1.72 (−6)]	[8.16 (−7)]	[1.72 (−6)]	[8.16 (−7)]	[1.72 (−6)]
Cub	1.05 (−6)	2.20 (−6)	2.34 (−6)	7.40 (−7)	5.64 (−7)	8.10 (−7)
	[1.04 (−6)]	[7.32 (−7)]	[5.55 (−7)]	[7.32 (−7)]	[5.55 (−7)]	[8.02 (−7)]
Aki	2.40 (−6)	3.80 (−6)	4.39 (−6)	2.40 (−6)	2.40 (−6)	2.40 (−6)
	[2.37 (−6)]	[2.37 (−6)]	[2.37 (−6)]	[2.37 (−6)]	[2.37 (−6)]	[2.37 (−6)]
But			3.58 (−6)			
			[1.26 (−6)]			
Geo			3.25 (−6)			
			[1.15 (−6)]			
Har			5.50 (−6)			
			[2.39 (−6)]			
Hym	6.42 (−7)	1.99 (−6)	2.10 (−6)	5.36 (−7)	2.93 (−7)	6.03 (−7)
	[6.42 (−7)]	[5.35 (−7)]	[2.93 (−7)]	[5.35 (−7)]	[2.93 (−7)]	[6.03 (−7)]

the modifications required for monotonicity degrade the solution at the extrema and improve it in the monotonic regions (second or third column versus first). Unlike the Hermite cubic, but like the rational quadratic interpolant, the rational cubic interpolant with C^1 continuity offers improvement over C^0 continuity (third column versus second), except with the Hyman and cubic estimates measured over the entire domain. For practical purposes C^0 and C^1 produce the same average error with the Hyman estimate. When the strict monotonicity condition is relaxed at the extrema so that the rational cubic interpolant relies on its convexity properties, the error is reduced further (fourth and fifth columns versus second and third, respectively). It is also of interest to note that, in this case, the error over the entire domain is almost the same as that within the monotonic domain (for example, compare bracketed with corresponding unbracketed terms in columns four and five). The extrema are no longer responsible for a larger fraction of the error. Column six (labeled NCC1) represents implementation

2 of the interpolant as discussed in §§ 2.2 and 3.2. We see that implementation 1 and 2 result in identical accuracy for the arithmetic and Akima derivative estimates that automatically satisfy NCC1. Their accuracy is almost equivalent for the cubic and Hyman derivative estimates, suggesting the particular implementation makes little difference.

3.3.4. Bernstein quadratic (Table 6). Within any column, the relative standings of the derivative estimates are the same as with the rational cubic (Table 5) with the exception of a reversal of the Fritsch–Butland and geometric estimates. These two, along with the harmonic and Akima approximations, have the largest errors and will not be considered further here. The errors from the Bernstein quadratic scheme are consistently higher than the corresponding ones from the second version of the rational cubic (Table 6 compared to the corresponding entry in Table 5). The only exception is the cubic derivative estimate with no limiter, where the errors are identical to the accuracy shown in Table 5.

TABLE 6
Error measure for the piecewise quadratic Bernstein polynomial interpolant, cosine bell shape, using 40-*point resolution. See Table* 2 *caption for an explanation of the numbers.*

Slope	No limiter	NCM0	NCM1	NCM0-EE	NCM1-EE	NCC1
Ari	2.18 (−6)	3.79 (−6)	2.98 (−6)	2.18 (−6)	1.10 (−6)	2.18 (−6)
	[2.16 (−6)]	[2.16 (−6)]	[1.09 (−6)]	[2.16 (−6)]	[1.09 (−6)]	[2.16 (−6)]
Cub	1.05 (−6)	2.40 (−6)	2.52 (−6)	7.86 (−7)	6.01 (−7)	1.00 (−6)
	[1.04 (−6)]	[7.77 (−7)]	[5.92 (−7)]	[7.77 (−7)]	[5.92 (−7)]	[9.94 (−7)]
Aki	2.51 (−6)	4.08 (−6)	4.65 (−6)	2.51 (−6)	2.51 (−6)	2.51 (−6)
	[2.47 (−6)]	[2.47 (−6)]	[2.47 (−6)]	[2.47 (−6)]	[2.47 (−6)]	[2.47 (−6)]
But			3.67 (−6)			
			[1.34 (−6)]			
Geo			3.75 (−6)			
			[1.56 (−6)]			
Har			6.46 (−6)			
			[3.16 (−6)]			
Hym	7.05 (−7)	2.23 (−6)	2.30 (−6)	6.26 (−7)	3.50 (−7)	8.42 (−7)
	[7.04 (−7)]	[6.26 (−7)]	[3.49 (−7)]	[6.26 (−7)]	[3.49 (−7)]	[8.41 (−7)]

The Bernstein quadratic scheme tends to have slightly smaller errors than the Hermite cubic with the Hyman and cubic derivative approximations when no limiters are applied to the derivatives (column one of Table 6 versus column one of Table 2) but the differences seem negligible. When the appropriate C^0 monotonic limiters are applied, the Hermite cubic tends to be the better of the two with these derivative estimates. The Bernstein quadratic interpolant tends to be better with the C^1 monotonic limiters (columns two and three of Table 6 versus columns two and three of Table 2, respectively). In short, neither the Bernstein quadratic nor Hermite cubic forms are consistently more accurate than the other.

3.4. Conclusions from the interpolation tests. In the previous section we have compared the various schemes in interpolating a well-resolved cosine bell shape. In [21] we make similar comparisons for well-resolved Gaussian and triangular test shapes, and for the three corresponding poorly resolved test shapes. Out of the mass of numbers considered in these comparisons there are logical inferences to be drawn relating the various schemes to each other. These conclusions may not be universal, as definite known properties of particular fields might be used to advantage in the interpolation scheme. Minor exceptions can be found in our tables that might imply some other scheme is ideal for such specific applications. It is also possible that our conclusions would change with a different error measure.

We begin by itemizing our conclusions regarding the interpolating functions:

- The Hermite cubic and the second version of the rational cubic interpolant appear to be the most useful interpolation formulas. The first version of the rational cubic interpolant is consistently inferior to the second.
- The Bernstein quadratic interpolant is generally of comparable accuracy to the rational form mentioned above. We found it to be somewhat more difficult to program for the various special cases, which results in a corresponding increase in the complexity of computer code and execution time.
- The rational quadratic interpolant is of comparable accuracy to the SCM limited Hermite cubic for monotonic data, but it does not allow the flexibility of the Hermite cubic near extrema, nor does it allow for the concave/convex structure provided by versions of the rational cubic interpolant. For data that have an extremum, this scheme is not recommended because there is no alternative to assuming the slope goes to zero at a discrete extremum. This results in much larger errors in the vicinity of the extremum than the cubic, rational cubic, and piecewise quadratic spline forms.

Conclusions regarding the derivative estimates follow:

- The geometric mean, harmonic mean, and Fritsch–Butland derivative estimates are consistently less accurate than the others. Their virtue is their simplicity. While they may result in visually pleasing interpolants, they are generally of insufficient accuracy for many applications. The rational-linear derivative estimate [9], equivalent to that suggested by McAllister and Roulier [17], and to the harmonic mean estimate suggested by Fritsch and Butland [7] for equally spaced data, is the least accurate of all the derivative estimates. The Fritsch–Butland derivative estimate is always more accurate than the rational linear estimate.
- The Akima approximation performs extremely well for data with small scale features, but less well for the broader, more rounded shapes. Careful examination of the results suggests the Akima scheme is actually quite accurate in the vicinity of the extrema, and much less accurate over the rest of the domain.
- Except for the intersection of straight lines such as triangular peaks where the Akima estimate shines, the Hyman derivative estimate is the most accurate, followed generally by the cubic, then arithmetic. The disadvantage of the cubic derivative approximation is that it does not provide for a C^1 continuous interpolant, whereas the others do.
- Monotonicity constraints generally improve the interpolation of monotonic data and data approaching a flat plateau. These constraints degrade the interpolation near extrema by not allowing any overshoot that might be implied in the underlying data. The derivative estimate is constrained to be zero in the vicinity of an extremum with the C^1 form. The C^0 continuity constraint is less serious in this than the C^1.
- Where strict monotonicity is not required, relaxation of the monotonicity condition at any extremum seems desirable to allow the interpolant to form an extremum somewhere other than at a data point. Application of Hyman's limiter for the Hermite cubic seems desirable to prevent overshooting in the approach to a flat or nearly flat plateau.

In general the accuracy of the interpolation does not vary by more than a factor of two or three between schemes. The exception to this statement occurs when the shape to be interpolated is analytic and resolution is high, in which case the high formal accuracy of the cubic interpolant, and the Hyman fourth-order derivative estimate result in substantial increases in accuracy over other schemes. We saw this demonstrated in the well-resolved Gaussian test shape, points.

We mention in passing that we also tested other C^0 sufficient conditions for monotonicity with the Hermite cubic interpolation form that modify the derivative estimates to lie on the elliptical boundary of $\mathcal{M}_{ns}(12)$ rather than to the more restrictive boundary of $\mathcal{M}_b$. These limiting forms involve extra calculations and result in minute but discernible improvements in the accuracy of the representation.

4. One-dimensional semi-Lagrangian advection. As mentioned in the Introduction, the primary motivation for this intercomparison was to improve the interpolation used in the technique identified in meteorology as semi- or quasi-Lagrangian advection [13], [14], [20], [19]. The technique has also been used in other fields [23], [11]. It is similar to a method suggested by Van Leer in his earlier papers on conservative monotonic advection schemes. It was abandoned by Van Leer because of the difficulty of *strictly* enforcing both conservation and monotonicity simultaneously. It also shares common characteristics with the pseudo-particle methods suggested in Takewake, Nishiguchi, and Yabe [26] and Takewake and Yabe [27].

In meteorology, the semi-Lagrangian technique has found increasing acceptance because of its accuracy and unconditional stability with respect to the size of the timestep. In practice, the community has found that when using nonshape-preserving polynomial interpolation, the solutions are of sufficient accuracy that they adequately conserve mass even without a strict enforcement of the conservation property. The solutions do however show evidence of a "ringing" effect generating negative (hence nonphysical) values of the advected quantity during an integration in time.

The initial motivation for this work was to reduce or eliminate these over- and undershoots by using shape-preserving interpolation functions in the semi-Lagrangian scheme. Using shape preserving interpolation the semi-Lagrangian integration scheme becomes either monotonic (see, e.g., [28], [29], [2]) in the sense that extrema will not increase due to the numerical method, or essentially nonoscillatory [10], in the sense that no new extrema are introduced in the solution by the method.

In this section some of the more accurate schemes examined in the previous section are retested in the simplest possible transport problem, the one-dimensional advection of a scalar by a constant advecting wind. The Eulerian form of the evolution equation for the advection of a scalar field in the absence of sources and sinks is

$$\frac{\partial f(x,t)}{\partial t} + v(x,t)\frac{\partial f(x,t)}{\partial x} = 0. \tag{34}$$

Here f is a scalar field, t is time, and v is the velocity. Given a distribution of f at time t, the solution at time $t+\delta t$ is

$$f(x, t+\delta t) = f(\hat{x}, t). \tag{35}$$

Here, it is assumed that v is constant in space and time, and the departure point becomes

$$\hat{x} = x - v\,\delta t. \tag{36}$$

When v is not constant, determining the trajectory is nontrivial; however, solution methods have been developed for use in geophysical problems. This paper will focus exclusively on the interpolation aspects of the solution. Given the true departure point, an interpolation is made to find the value there, $f(\hat{x}, t)$. From the interpolated values of f, the solution to (35) follows trivially. Thus, semi-Lagrange transport with a constant advection velocity is equivalent to "recursive interpolation." This is of course a tremendous simplification of our actual problem of interest, but provides a simple meaningful test of the efficacy of the approach. Our actual interest is in problems involving nonconstant velocity fields, nonuniform grids, and multiple dimensions [30].

Many different nonshape-preserving polynomial interpolation schemes have been used over the years for semi-Lagrangian transport, the simplest being linear interpolation. Semi-Lagrangian transport then becomes very similar to the familiar "upwind" differencing used in solving advection equations; it automatically provides a monotonic solution, but is also extremely diffusive. Quadratic interpolation schemes were introduced into meteorological problems in the late 1950s and early 1960s. More recently, cubic, cubic spline, and deficient cubic spline interpolation methods have been used [20], [14], [22], [19]. The single example of the use of shape-preserving interpolation is found in [23], but these tests used an extremely high resolution, and did not provide the objective statistics needed to intercompare results. Akima's scheme was tested by Huffenus and Khaletzky [11] in a semi-Lagrangian scheme and found to be quite accurate. While this interpolation is not monotonic, and hence does not produce a positive semidefinite solution to (34), the oscillations are much reduced over a regular cubic formulation.

We reduce the number of interpolation schemes to test by considering only those that rated well in the interpolation tests or have some historical importance. Thus a discussion of only the highly scoring and more readily applicable Hermite and rational cubic version 2 interpolants appears here. The quadratic Bernstein polynomial interpolant rates near the rational cubic scheme but is more complicated to apply. Thus we do not consider it here. Only the more accurate arithmetic, cubic, Hyman, and Akima derivative estimates with appropriate limited and unlimited forms are tested.

The advection equation is integrated by choosing $v\,\delta t/\delta x = \pi/12$. An irrational number is chosen so that the relative position of the shape and gridpoints change with time. Given a long enough integration it becomes almost uniformly distributed over the domain. For the following figures the coordinate system of perspective moves with the advecting velocity. The equation is integrated for 1,000 timesteps with snapshots of the evolution of the scalar field plotted every 200 timesteps. The line code in the figures is such that the shorter the pattern members, the later in time the solution. The scalar field deforms slowly due to errors in the interpolation. During some of the integrations, the scalar field stops deforming after a while and propagates with a characteristic shape for the remainder of the integration. This suggests that there may be preferred shapes for given interpolation schemes that are handled very accurately.

We use cosine bell and square wave initial conditions, with 11 nonzero points on the interval. These shapes are relatively narrow, and it might be expected that the schemes will be ordered in accordance with the poorly resolved test case interpolation results discussed in [21] and summarized in § 3.

4.1. Square wave. Figure 1 shows the results of integrations of the unlimited forms of the arithmetic, cubic, Hyman, and Akima derivative estimates with the Hermite cubic interpolant. The left panel shows snapshots of the evolution of the shape of the scalar field, with a line pattern as described above. The right panel shows the evolution of the normalized mean, maximum, minimum, sum of negative values, excess greater than one, and mean absolute error. These quantities appear within the panel ordered approximately from top to bottom. Two lines appear for each quantity. The thinner line shows the "ideal" behavior. The heavier line shows the "actual" behavior. Occasionally,

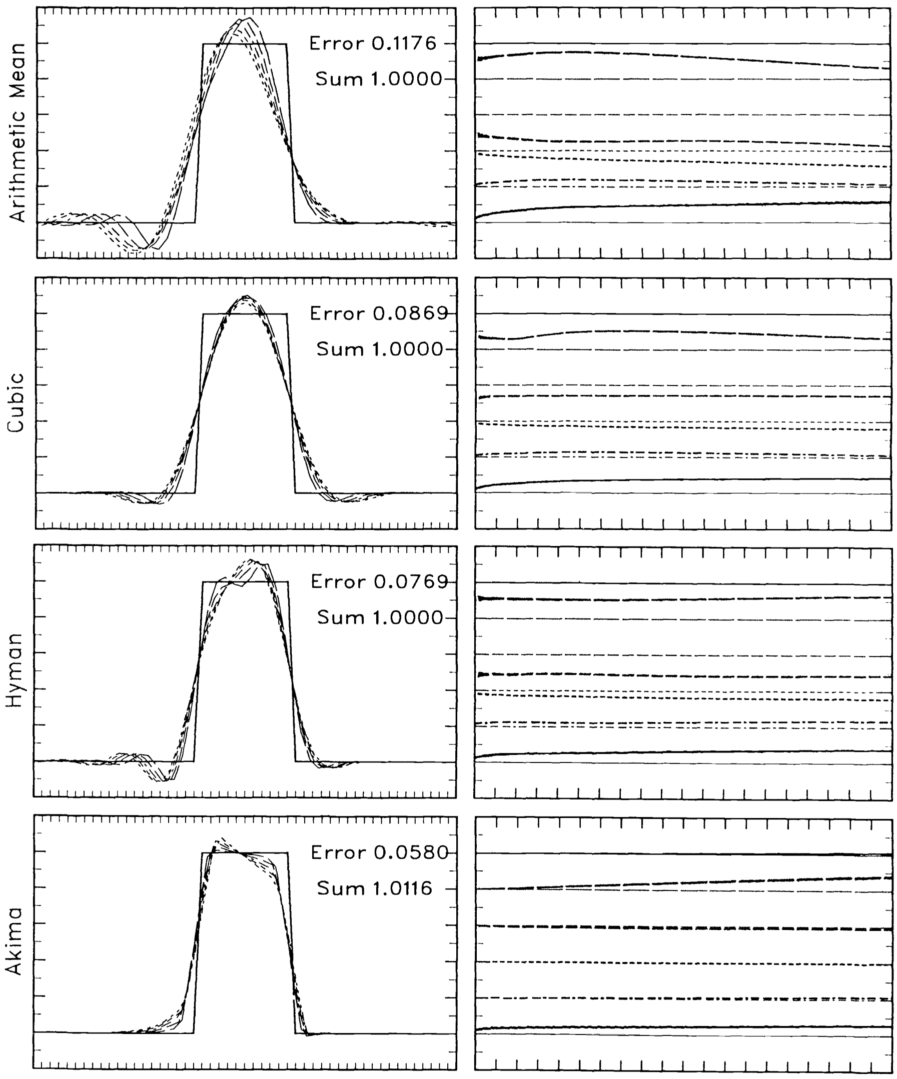


FIG. 1. *Hermite cubic interpolant, square wave initial conditions, unlimited form of the derivative estimates. The left panels are snapshots of the evolution of the shape with time (every* 200 *timesteps); the shorter the line segment, the later in the integration (i.e., solid line—original shape; dotted line—$t = 1000\delta t$). Printed numbers give the normalized error and mean after* 1,000 *timesteps. The right panels are the diagnostics of the evolution with respect to time, from top to bottom. Upper solid—normalized mean value; long dashed—normalized maximum value; medium dashed—normalized minimum value; short dashed—normalized sum of value less than zero; dash dot—normalized sum of values greater than one; lower solid line—*RMS *error. All values are plotted with a horizontal line at the ideal value, using the same line pattern. The ordinate ticks represent intervals of* 0.1.

the lines are coincident, so that only one appears on the panel in which case the scheme does very well according to that measure. The numbers appearing in the upper right-hand corner of the panels on the left are the final value of the normalized error, and mean. Each tick mark on the ordinate represents a 0.1 variation.

Strong oscillations (less than 10 percent of the true signal) usually associated with dispersion errors are evident in the Hyman and arithmetic derivative estimates. The cubic derivative estimate shows less over/undershoot and the Akima estimate the least. The Akima version seems to be slowly evolving towards a peaked shape. As mentioned before, strict mass conservation is not a priori maintained by semi-Lagrangian schemes, rather mass would be preserved by accurate solution of the evolution equation. The total mass (upper solid lines of right panels, and "Sum") remains constant to two percent over the 1,000 timesteps for all the schemes with the Akima scheme showing the largest increase in mass (1.2 percent). There is a substantial component of the field for the first three schemes consisting of negative values. If we neglect this negative component of the field (on the grounds that it can have no physical basis) when computing the integral of the field, the conservation of mass for these schemes is much worse. There is also a substantial component of the field that is greater than one. With the arithmetic and cubic estimates, this overshoot is largest early in the forecast and then decreases. With the Hyman estimate, these excess values set up early in the forecast then shift relative to the overall structure. With the Akima estimate, they increase with time; however, the Akima estimate shows the least tendency for over/undershooting error in the unlimited form. There is also little tendency for negative values with the Akima scheme. In fact, the Akima estimate solutions are the most accurate, followed by those solutions that use the Hyman, cubic, and arithmetic derivative estimates.

Solutions that arise by applying SCM0 limiters for the Hermite cubic interpolant appear in Fig. 2. The monotonicity condition has improved the solution by eliminating the overshoot at the expense of an increase in the diffusion and decrease in the maximum value, at least for the arithmetic and cubic derivative estimate versions. The Hyman and Akima derivative estimate versions are improved by the limiting procedure, with little or no increase in diffusion of the shape. The error in the mass conservation has increased for the less accurate forms (arithmetic and cubic), but there is very little difference for the two more accurate forms (Akima and Hyman). We also note that the mass loss is less than the mass associated with the over- and undershoot of the unlimited forms. The SCM0 limited forms show slightly less overall accuracy when compared to the corresponding unlimited forms, but the difference is small for the Hyman estimate.

Imposition of the SCM1 constraint results in a very substantial increase in the error, which can be seen to be due to error in the phase of the solution, with the exception of the cubic derivative estimate scheme (Fig. 3). However, it must be remembered that the cubic estimate is C^1 only where the derivatives were modified by the limiting procedure. Elsewhere it remains C^0. The total mass tends to increase slightly with the C^1 forms, whereas it decreased slightly with the C^0 forms. This phase error is also seen with the Hyman limited form (Fig. 4) where overshooting is engendered by the slight relaxation of the monotonicity constraint in the vicinity of extrema, and a substantial increase in error in the mass conservation is evident.

Finally, Fig. 5 shows the application of only the necessary condition for monotonicity with C^0 continuity (NCM0) rather than the sufficient condition (SCM0). This seems sufficient to remove the unwanted oscillatory behavior in the Hermite interpolant with the Hyman and Akima derivative estimates although as expected it is not sufficient to guarantee monotonic solutions. These schemes are slightly more accurate than the corresponding SCM versions, and slightly less accurate than the unlimited versions. It is interesting to note that the C^1 version of this limiter (not shown) results in a substantial increase in the over/undershooting of the solution for these derivative estimates. The nonlinear Akima derivative estimate on the other hand is not sensitive to the difference between the C^0 and C^1 estimates.

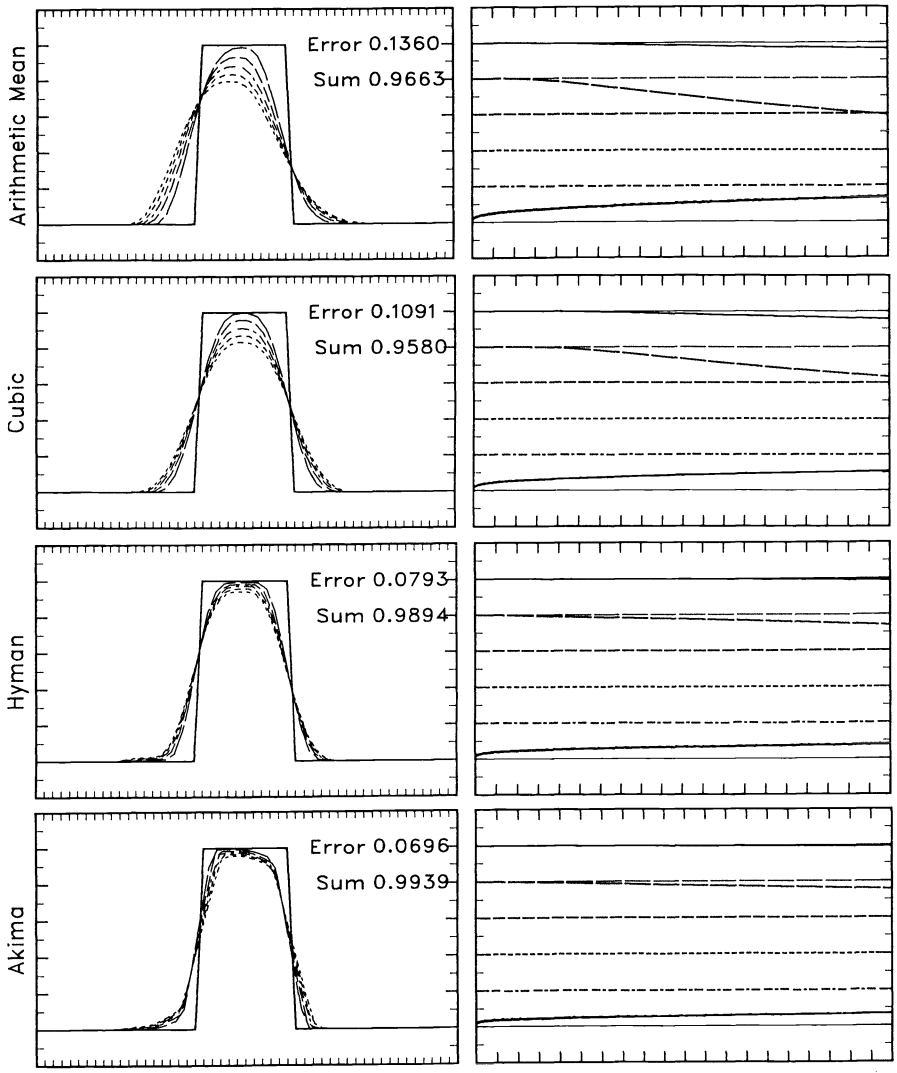


FIG. 2. *Hermite cubic interpolant, square wave initial conditions,* SCM0 *limited form of the derivative estimates. See Fig.* 1*'s legend for an explanation of the panels.*

Figures 6 and 7 show forecasts made using the second version of the rational cubic interpolant. As may be seen from Fig. 6, compared to Fig. 1, the rational interpolant produces a more rounded profile for the unlimited derivative estimates along with a substantial reduction in the over/undershooting, and less dispersion, compared with the Hermite form of the interpolant, for the arithmetic, cubic, and Hyman derivative estimates. The Hyman estimate is slightly more accurate than the corresponding solution that uses the Hermite interpolant; all others are slightly less accurate. Overall we consider the unlimited rational interpolant solution to be better than the unlimited Hermite. The exception is the solution using the Akima derivative estimate that is worse with the rational than the Hermite polynomial interpolant.

Use of the C^1 necessary condition for monotonicity with the rational cubic interpolant (Fig. 7) increases the diffusion slightly for the polynomial derivative estimates, but not for the Akima estimate. The C^0 version is almost identical (not

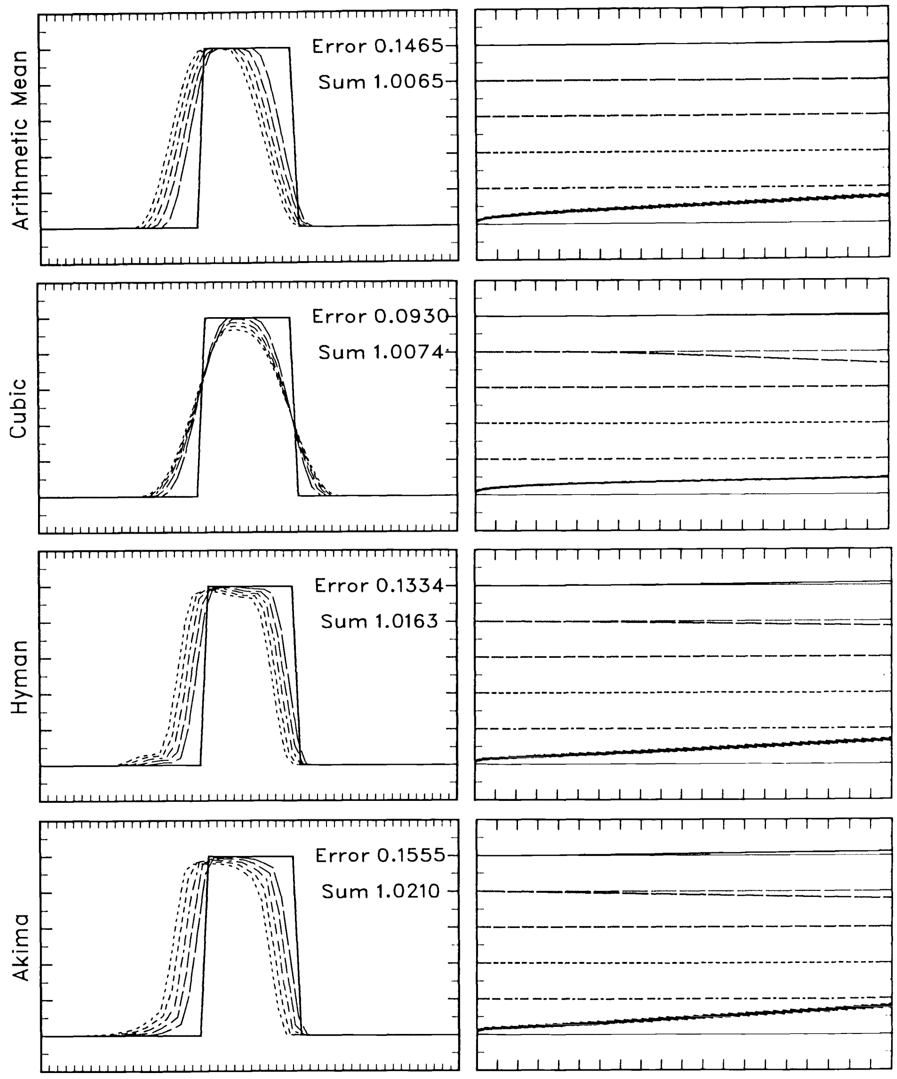


FIG. 3. *Hermite cubic interpolant, square wave initial conditions,* SCM1 *limited form of the derivative estimates. See Fig.* 1*'s legend for an explanation of the panels.*

shown). As expected, the overshooting is eliminated. Unlike the Hermite interpolant (see Fig. 3 or 4), the C^1 monotonic rational interpolant does not show an increase in the phase error, when compared to a C^0 version. In addition, the total mass decreases slightly with the C^0 and C^1 versions of the rational interpolant as the C^0 Hermite did but unlike the C^1 Hermite form with which it increased slightly. For the C^1 forms the rational cubic with the Hyman derivative estimate limited to satisfy NCM1 is clearly the best.

4.2. Cosine bell. The strongly oscillatory results seen in the unlimited Hermite interpolant square wave tests also appear in the cosine bell results (Fig. 8). The Akima derivative estimate shows the least error followed by the Hyman estimate. The Akima estimate has evolved to an almost triangular form leading the true solution and the Hyman shows undershooting. There is also evidence (right panel) of a $2\delta t$ oscillation in the maximum amplitude of the peak with the Akima estimate, and the mass

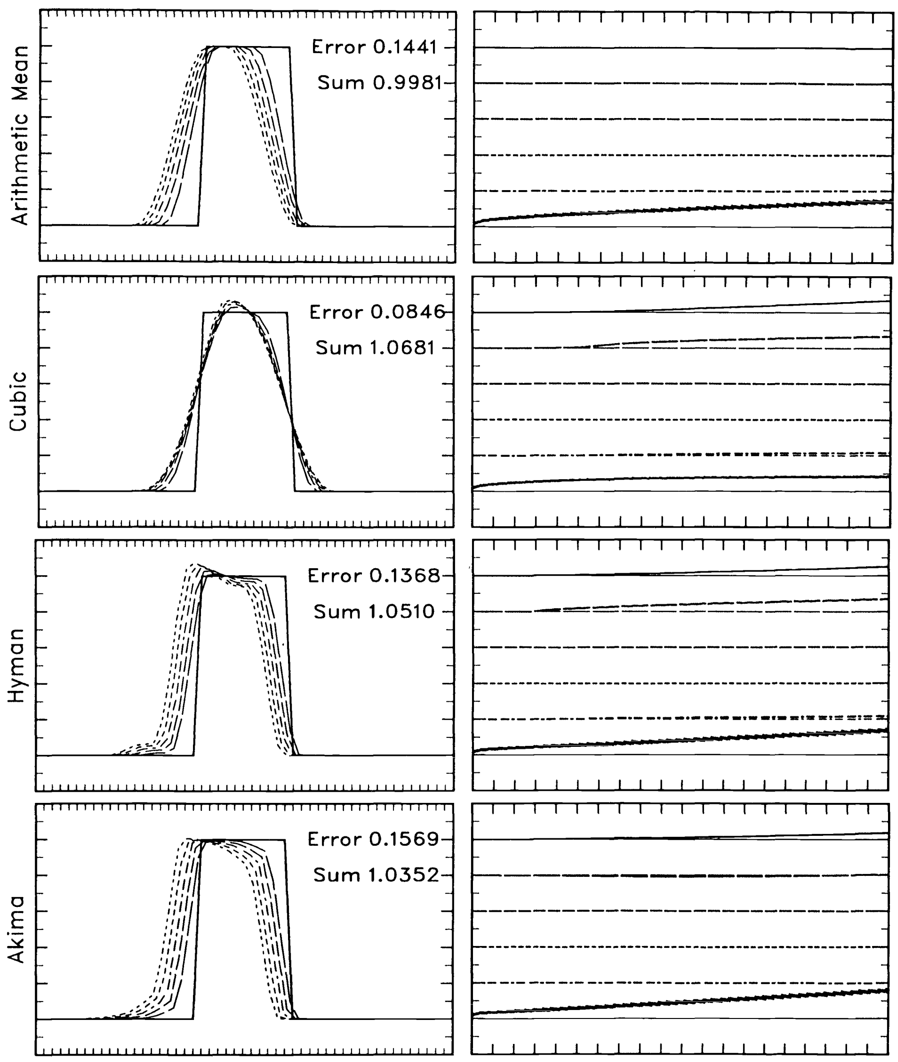


FIG. 4. *Hermite cubic interpolant, square wave initial conditions, Hyman C^1 limited form of the derivative estimates. See Fig. 1's legend for an explanation of the panels.*

conservation is worse for the Akima version than that using other derivative estimates. The cubic and arithmetic mean derivative estimates are much less accurate, and clearly less satisfactory.

The impact of the SCM0 limiting on the extremum value is much more pronounced for the cosine bell than was seen in the square wave. This is a familiar result with monotonic transport schemes where "clipping" will inevitably take place when the peak value moves between two data points. It can be seen in Fig. 9 for the SCM0 form, and occurs consistently for other limiters as well. The amplitude has been reduced to half its original value over the 1,000 timesteps for all interpolation schemes that limit extrema. The C^1 limiters (not shown) also produce the very strong phase error seen in the square wave tests. Errors for the more accurate Akima and Hyman forms have approximately doubled, and mass conservation is very much worse than with the unlimited forms.

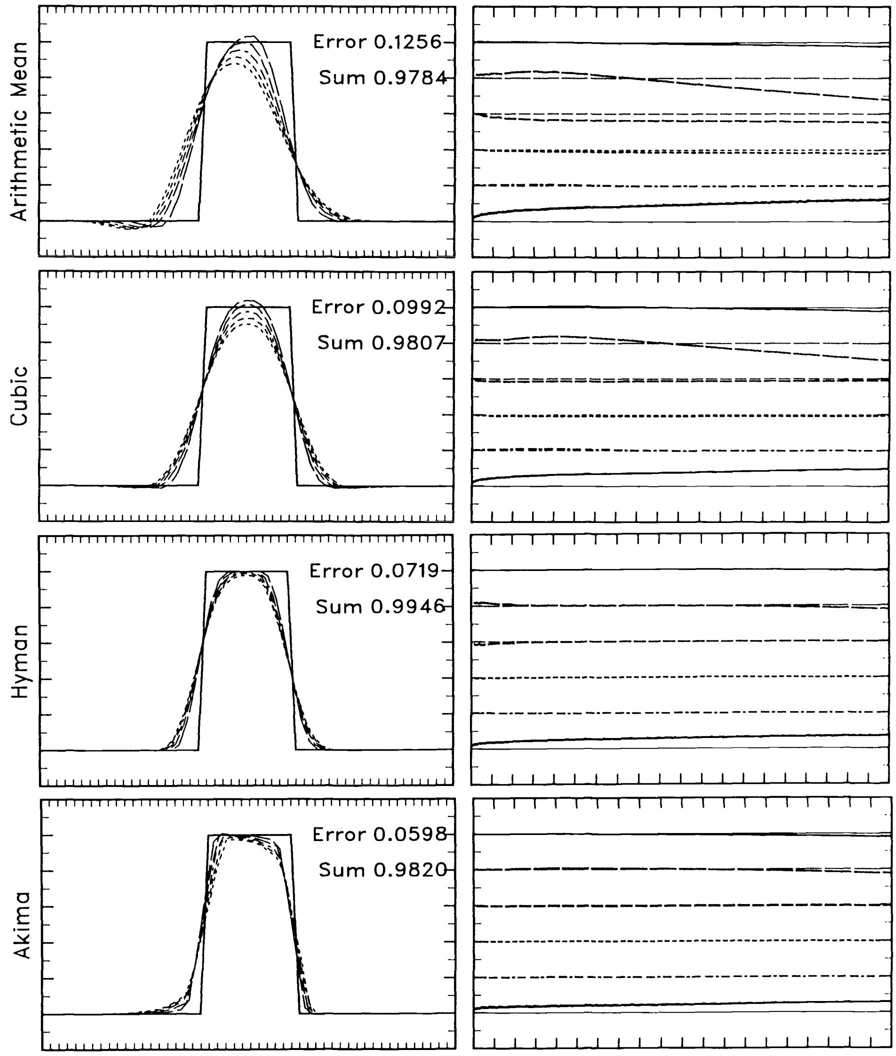


FIG. 5. *Hermite cubic interpolant, square wave initial conditions,* NCC0 *limited form of the derivative estimates. See Fig.* 1'*s legend for an explanation of the panels.*

The unlimited forms utilizing the rational cubic interpolant version 2 (Fig. 10) are, in general, more strongly damped than the unlimited Hermite interpolant forms. The form using the Hyman interpolant is the most accurate of the rational interpolants, with about the same accuracy as the corresponding Hermite form. There is very little undershoot, and the maximum value has decreased to about 75 percent of its original value. It has preserved the shape of the initial condition very well except for the damping. Application of the necessary condition for monotonicity (NCM1, not shown) increases the damping to 40 percent of its original value unless it is excluded from application at the extrema, in which case it leaves the interpolation essentially unaffected. Use of the necessary condition for convexity/concavity (NCC1) increases the error and conservation error slightly compared to the unlimited forms, but provides a shape-preserving, essentially nonoscillatory solution (Fig. 11). These errors are much smaller than the shape-preserving forms seen using the Hermite interpolants.

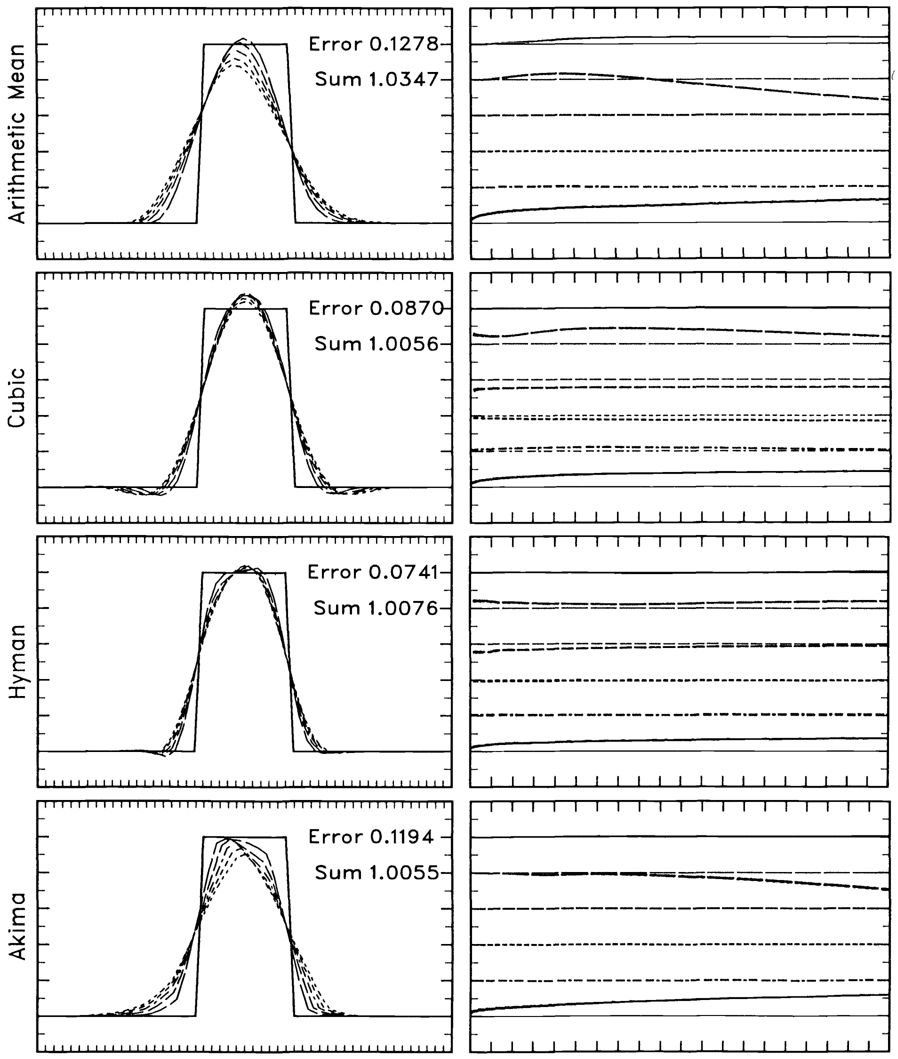


FIG. 6. *Rational cubic interpolant, square wave initial conditions, unlimited form of the derivative estimates. See Fig.* 1'*s legend for an explanation of the panels.*

5. Summary and conclusions. A large number of shape-preserving (and some non-shape-preserving) interpolants were compared to examine their relative accuracy. The large number of schemes arise by considering combinations of interpolating forms (piecewise Hermite cubic, three forms of piecewise rational, and Bernstein polynomials with extra knots), methods of estimating the derivatives (Akima, Hyman, arithmetic, harmonic, and geometric mean, and Fritsch–Butland) and derivative constraints that provide for shape preservation suggested by Carlson and Fritsch [3], de Boor and Swartz [4], and Hyman [12].

The intercomparisons were made by first examining the ability of the schemes to interpolate a set of evenly spaced data drawn from three test shapes. Two resolutions were used to examine the sensitivity of the interpolation schemes to data density. Results for the well-resolved cosine bell are documented in § 3.3, and all others in [21]. The schemes were evaluated in terms of their accuracy over the interval in which

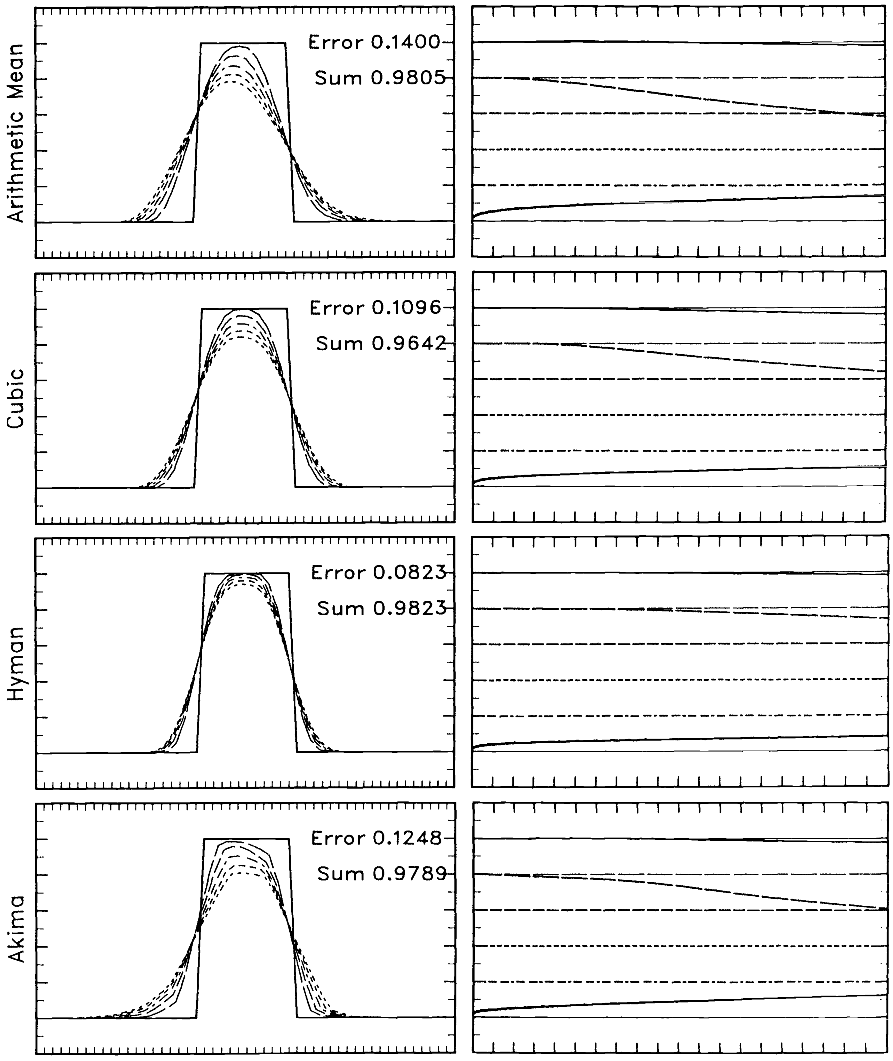


FIG. 7. *Rational cubic interpolant, square wave initial conditions,* NCC1 *limited form of the derivative estimates. See Fig.* 1'*s legend for an explanation of the panels.*

the data were monotonic, and the interval containing an extremum. The Hermite cubic and rational cubic (version 2) interpolants appear to be the most useful of the interpolants. The Bernstein polynomial approach is nearly as accurate, but is somewhat more complex to code. When tested with shapes well resolved by the data (shapes resolved by 40 points), the Hyman, cubic, and arithmetic mean derivative estimates provide the highest accuracy, in descending order. When tested with relatively poorly resolved shapes with sharp gradients, defined by 11 nonzero valued data points, the ranking changes. The Akima derivative estimate stands out when used with the Hermite cubic polynomial interpolant, and the Hyman estimate appears best when used with the forms of the rational cubic interpolant.

In general, the interpolants that strictly preserve monotonicity have large errors in the vicinity of an extremum. These errors arise because the interpolant is forced to have its extrema at data points while the underlying data may imply extrema between

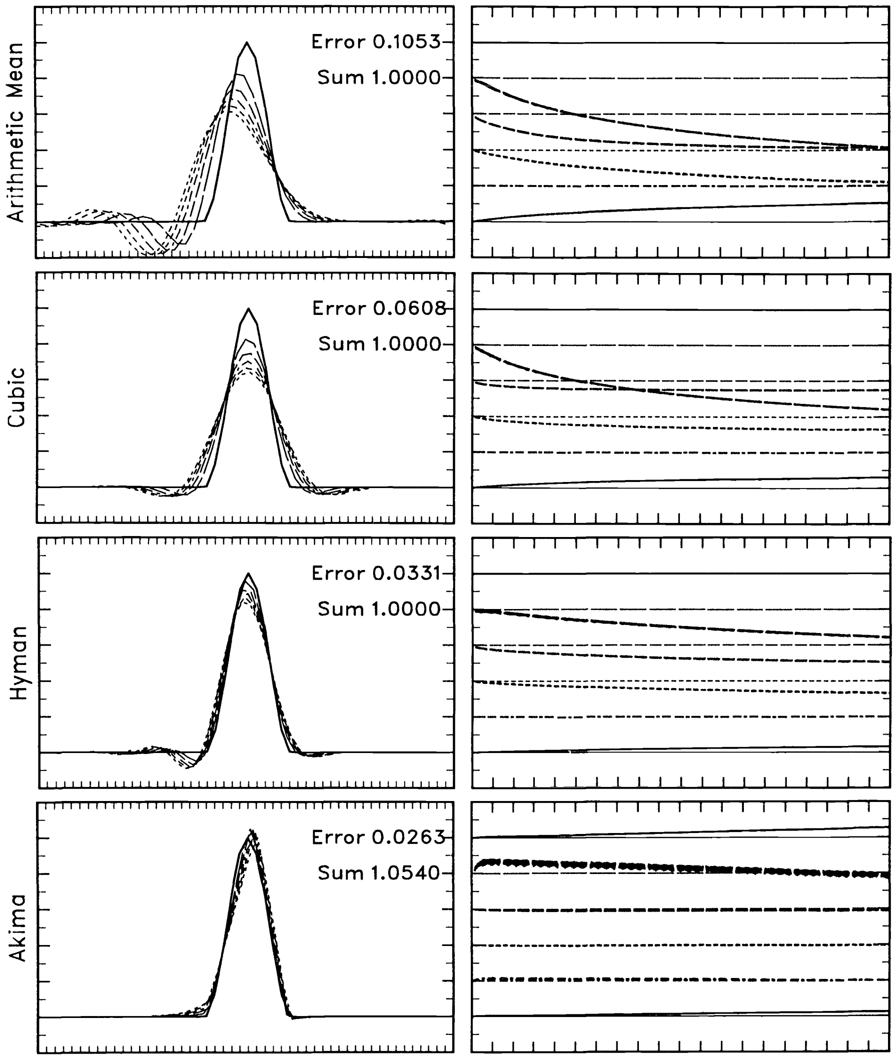


FIG. 8. *Hermite cubic interpolant, cosine bell initial conditions, unlimited form of the derivative estimates. See Fig.* 1'*s legend for an explanation of the panels.*

data points. Some form of relaxation of a strict monotonicity constraint is required for reasonable accuracy near extrema. We have considered the less restrictive restraint suggested by Hyman, and the complete relaxation of a monotonicity constraint in the vicinity of extrema. We have also considered convexity constraints where appropriate. These techniques improve the accuracy by allowing possible overshooting.

We then compared the more favorable schemes via a test more closely allied to our particular interests, using shape preserving interpolation as part of the numerical solution to a one-dimensional advection problem solved by the semi-Lagrangian technique. Only the more accurate interpolants as determined from the first set of tests were considered. The test consisted of the advection of a scalar field by a constant velocity field with square wave and cosine bell shapes for initial conditions.

In general, the rational form showed much less over- and undershooting than the Hermite interpolant when the derivative approximations are unconstrained. The Hyman

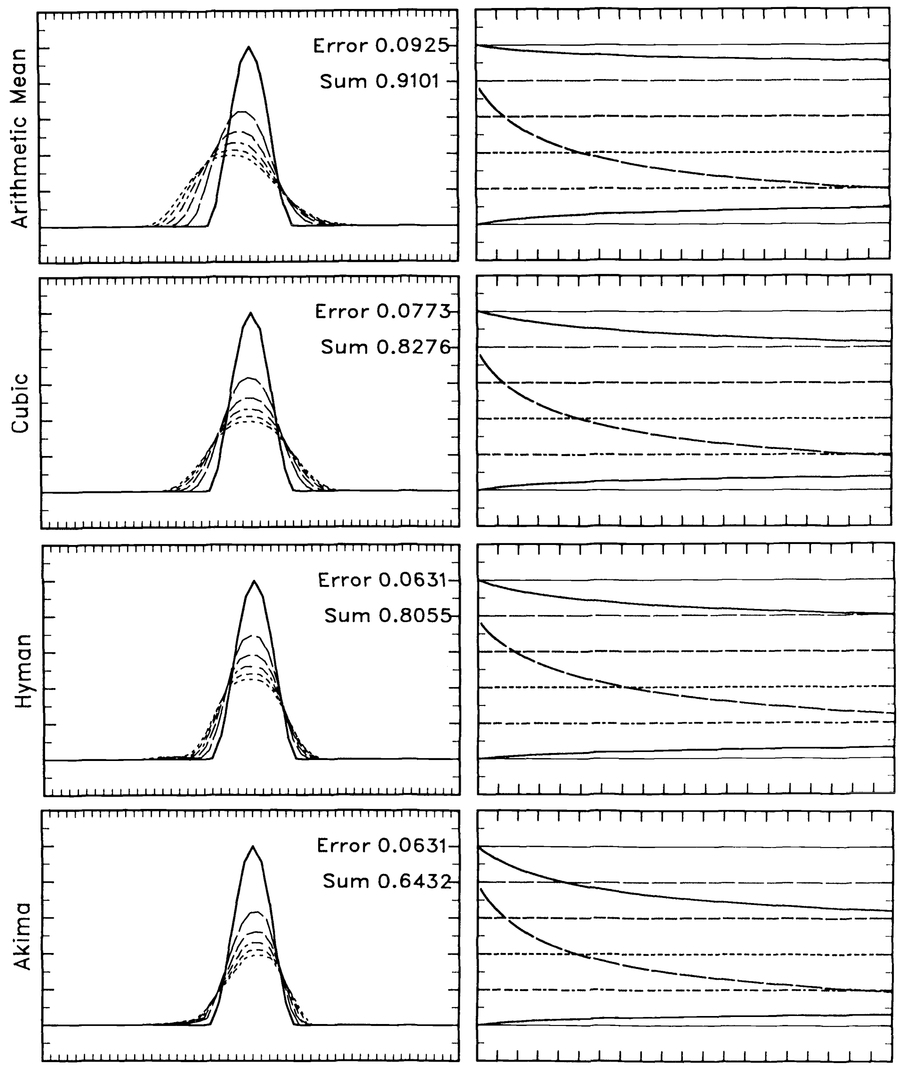


FIG. 9. *Hermite cubic interpolant, cosine bell initial conditions.* SCM0 *limited form of the derivative estimates. See Fig.* 1*'s legend for an explanation of the panels.*

and Akima derivatives constrained to satisfy the SCM0 are the most accurate monotonic form of the Hermite interpolant. They result in slight damping with the square wave and minimal diffusion. Significant phase error is introduced when the derivatives are constrained to satisfy SCM1. With the rational cubic interpolant, when the derivatives are constrained to satisfy NCM0, NCM1, or NCC1, the Hyman derivative approximation is best, comparing favorably with the best Hermite solutions. The Akima estimate is unacceptable with the rational cubic form. Unlike the Hermite cubic, the rational cubic does not show any increased phase error with C^1 limited derivatives compared to C^0. Thus if C^1 continuity is desired, the rational form should be used.

Application of the monotonicity conditions to the derivative estimates produces damping of the solution, especially with poorly resolved structures, by not allowing any overshoot implied by the data. The schemes that relax the monotonicity constraint somewhat, and hence are more closely allied to the essentially nonoscillatory schemes

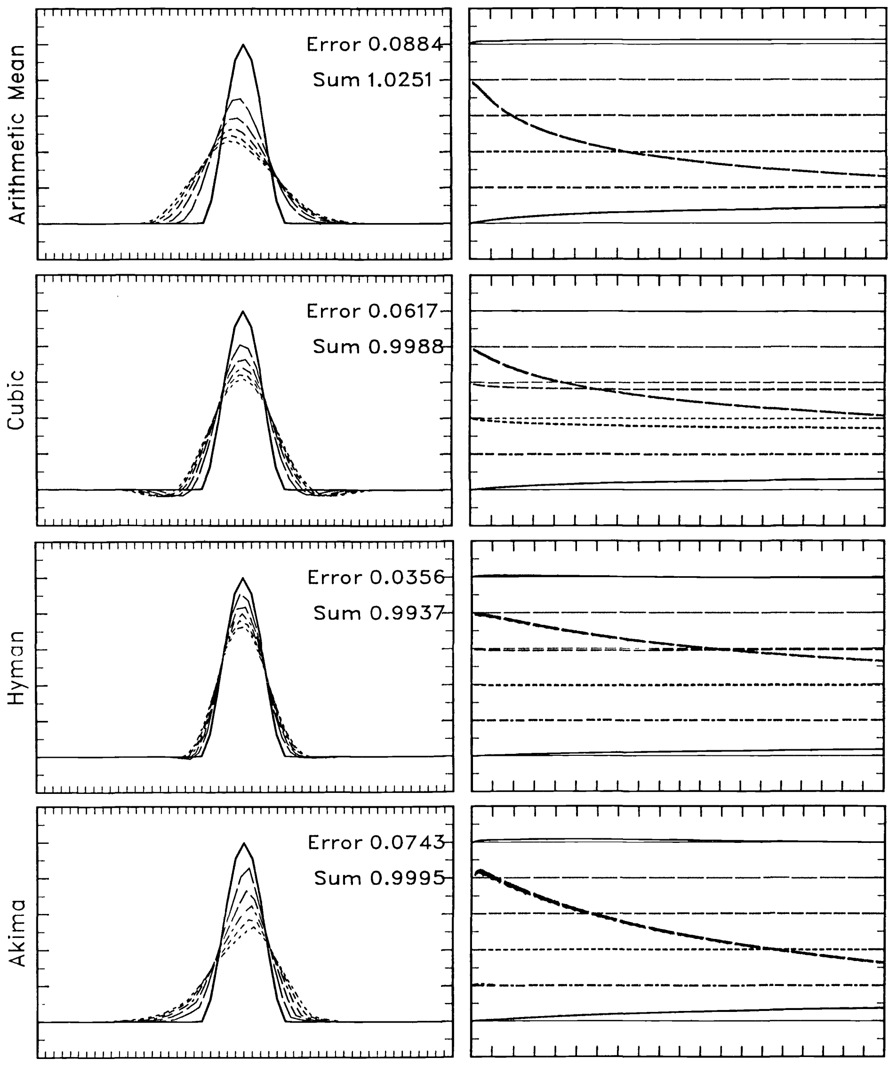


FIG. 10. *Rational cubic interpolant, cosine bell initial conditions. Unlimited form of the derivative estimates. See Fig.* 1'*s legend for an explanation of the panels.*

of Harten and Osher [10] are more accurate, because they do not entail this clipping phenomenon.

With no derivative modification, the Akima derivative approximation coupled with the Hermite cubic interpolant, which looks formally substantially less accurate than some of the other combinations tested, has the least mean absolute error of all schemes in the semi-Lagrangian integration test. Peak values were well maintained, and the shape was reasonably well preserved with a slow tendency to evolve toward a triangular shape. There was little overshooting of the true shape and the overshoot was confined to a few gridpoints. This result conflicts somewhat with that of Long and Pepper [14]. They used a two-dimensional test with a very narrow cosine hill going to zero over four gridpoints. This conflict may possibly result from the extremely short spatial scales used in their work, and the somewhat larger scales used here. The Long and Pepper [14] result is also at odds with Huffenus and Khaletzky [11], who (using

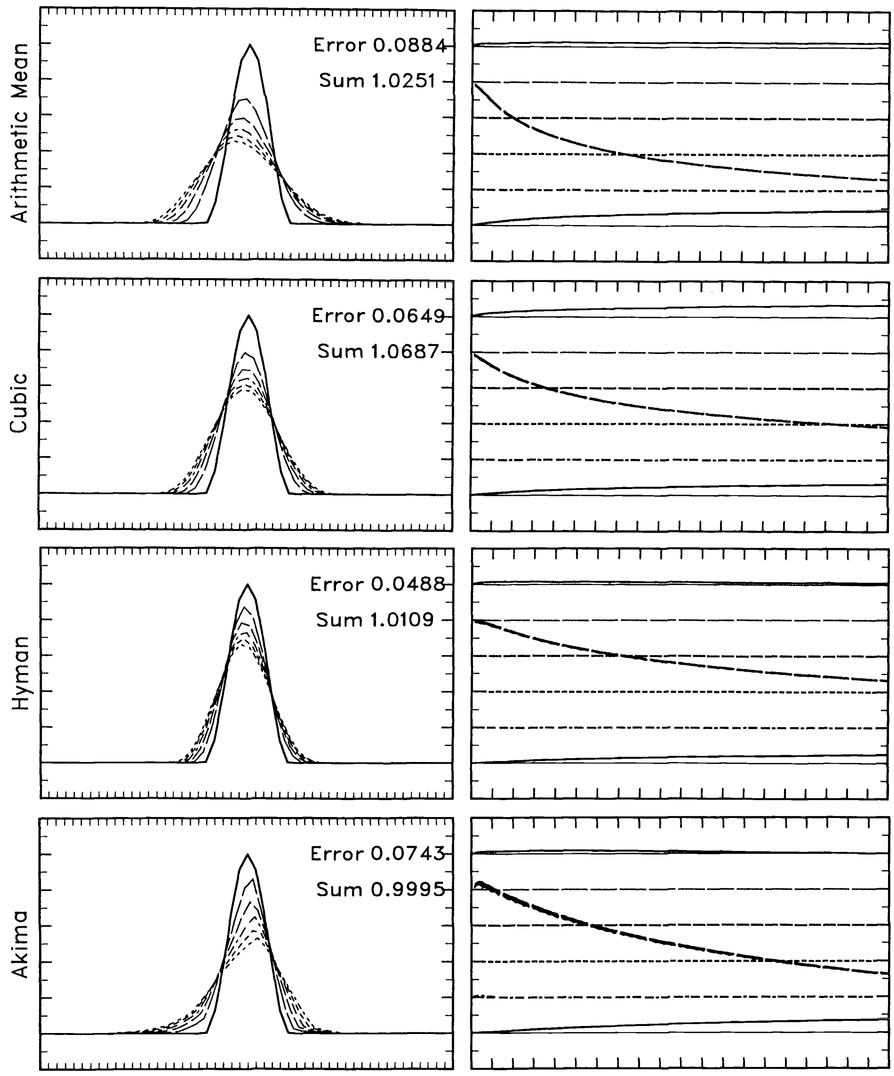


FIG. 11. *Rational cubic interpolant, cosine bell initial conditions.* NCC1 *limited form of the derivative esitmates. See Fig.* 1'*s legend for an explanation of the panels.*

a Gaussian test shape, with a half width of ≈3.1 grid intervals) found the Akima method to perform very well.

The other scheme that ranked quite well according to the semi-Lagrangian test was the rational cubic interpolant scheme with derivative estimates determined by the Hyman Algorithm. The scheme tended to damp the strongly peaked cosine bell to about 75 percent of its initial amplitude over the 1,000 timestep test, less than any other of the shape-preserving schemes. The shape remained substantially more coherent than other forms utilizing a rational interpolant.

There is an inevitable tradeoff between the simplicity of the Hermite cubic interpolant coupled with derivative estimates that automatically satisfy sufficient conditions for monotonicity but sacrifice accuracy, and the schemes that require more sophisticated derivative estimates with additional shape-preserving constraints, or more complex spline forms, or both. It is our experience that the benefits of the more complex algorithms are worthwhile for any problem that requires more than "visually pleasing" results. We have compared the various approaches with a variety of test cases, both for pure interpolation and coupled with semi-Lagrangian transport. The scope of these tests is necessarily limited. Other error measures could be adopted, and the question of unequally spaced data has not been addressed. According to our evaluations, there is no one best scheme for all test cases, although a few consistently appeared among the best. In specific applications, definite known properties of particular fields might be used to advantage in the interpolation scheme. The choice of a scheme from among the best here for a particular application should take such specific characteristics into account.

Acknowledgments. We thank E. Boettner for editorial assistance and for preparing the final manuscript and M. Pappas for assistance in drafting figures.

REFERENCES

[1] H. AKIMA, *A new method of interpolation and smooth curve fitting based on local procedures*, J. Assoc. Comput. Mach., 17 (1970), pp. 589-602.

[2] J. P. BORIS AND D. L. BOOK, *Solution of continuity equations by the method of flux-corrected transport*, in Methods in Computational Physics, Vol. 16, Academic Press, New York, 1977.

[3] R. E. CARLSON AND F. N. FRITSCH, *Monotone piecewise bicubic interpolation*, SIAM J. Numer. Anal., 22 (1985), pp. 386-400.

[4] C. DE BOOR AND B. SWARTZ, *Piecewise monotone interpolation*, J. Approx. Theory, 21 (1977), pp. 411-416.

[5] R. DELBOURGO AND J. A. GREGORY, *C^2 rational quadratic spline interpolation to monotonic data*, IMA J. Numer. Anal., 3 (1983), pp. 141-152.

[6] ———, *Shape preserving piecewise rational interpolation*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 967-976.

[7] F. N. FRITSCH AND J. BUTLAND, *A method for constructing local monotone piecewise interpolants*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 300-304.

[8] F. N. FRITSCH AND R. E. CARLSON, *Monotone piecewise cubic interpolation*, SIAM J. Numer. Anal., 17 (1980), pp. 238-246.

[9] J. GREGORY AND R. DELBOURGO, *Piecewise rational quadratic interpolation to monotonic data*, IMA J. Numer. Anal., 2 (1982), pp. 123-130.

[10] A. HARTEN AND S. OSHER, *Uniformly high-order accurate nonoscillatory schemes*, SIAM J. Numer. Anal., 24 (1987), pp. 279-309.

[11] H. HUFFENUS AND D. KHALETZKY, *The Lagrangian approach of advective term treatment and its application to the solution of Navier-Stokes equations*, Internat. J. Numer. Methods Fluids, 1 (1981), pp. 365-387.

[12] J. M. HYMAN, *Accurate monotonicity preserving cubic interpolations*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 645-654.

[13] T. N. KRISHNAMURTI, *Numerical integration of primitive equations by a quasi-Lagrangian advection scheme*, J. Appl. Meteor., 1 (1962), pp. 508-521.

[14] P. E. LONG AND W. W. PEPPER, *An examination of some simple numerical schemes for calculating scalar advection*, J. Appl. Meteor., 20 (1981), pp. 146-156.

[15] D. F. MCALLISTER, E. PASSOW, AND J. A. ROULIER, *Algorithms for computing shape preserving spline interpolations to data*, Math. Comp., 31 (1977), pp. 717-725.

[16] D. F. MCALLISTER AND J. A. ROULIER, *Interpolation by convex quadratic splines*, Math. Comp., 32 (1978), pp. 1154-1162.

[17] ———, *Algorithms for computing shape preserving osculatory quadratic splines*, ACM Trans. Math. Software, 7 (1981), pp. 331-347.

[18] A. MCDONALD, *Accuracy of multiply upstream semi-Lagrangian advective schemes*, Mon. Wea. Rev., 112 (1984), pp. 1267-1275.

[19] J. PUDYKIEWICZ AND A. STANIFORTH, *Some properties and comparative performance of the semi-Lagrangian method of Robert in the solution of the advection-diffusion equation*, Atmos. Ocean, 22 (1984), pp. 283-308.

[20] D. K. Purnell, *Solution of the advective equation by upstream interpolation with a cubic spline*, Mon. Wea. Rev., 104 (1976), pp. 42–48.

[21] P. J. Rasch and D. L. Williamson, *A comparison of shape preserving interpolators*, NCAR Tech. Note NCAR/TN-339+STR, National Center for Atmospheric Research, Boulder, CO, 1989.

[22] H. Ritchie, *Application of a semi-Lagrangian integration scheme to the moisture equation in a regional forecast model*, Mon. Wea. Rev., 113 (1985), pp. 424–435.

[23] M. M. Shoucri, *Numerical calculation of discontinuities by shape preserving splines*, J. Comput. Phys., 49 (1983), pp. 334–341.

[24] P. K. Smolarkiewicz, *A fully multidimensional positive definite advection transport algorithm with small implicit diffusion*, J. Comput. Phys., 54 (1984), pp. 325–362.

[25] H. Spath, *Exponential spline interpolation*, Computing, 4 (1969), pp. 225–233.

[26] H. Takewake, A. Nishiguchi, and T. Yabe, *The cubic-interpolated pseudo particle* (CIP) *method for solving hyperbolic-type equations*, J. Comput. Phys., 61 (1985), pp. 261–268.

[27] H. Takewake and T. Yabe, *The cubic-interpolated pseudo particle* (CIP) *method*: *Application to nonlinear and multi-dimensional hyperbolic equations*, J. Comput. Phys., 70 (1987), pp. 355–372.

[28] B. van Leer, *Towards the ultimate conservative finite difference scheme*, II. *Monotonicity and conservation combined in a second order scheme*, J. Comput. Phys., 14 (1974), pp. 361–376.

[29] ———, *Towards the ultimate conservative finite difference scheme*, IV. *A new approach to numerical convection*, J. Comput. Phys., 23 (1977), pp. 276–298.

[30] D. L. Williamson and P. J. Rasch, *Two-dimensional semi-Lagrangian transport with shape preserving interpolation*, Mon. Wea. Rev., 117 (1989), pp. 102–129.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 4, pp. 688-701, July 1990

BORDERED MATRICES, SINGULAR SYSTEMS, AND ERGODIC MARKOV CHAINS*

KING-WAH ERIC CHU†

Abstract. Systems of linear equations involving submatrices of singular M-matrices have been considered in the calculation of stationary distribution vectors of ergodic Markov chains. In this paper, an alternative appproach is suggested, using bordered matrices instead, motivated by the stability analysis of the first approach. The conditioning of this approach, which applies to general as well as M-matrices, is considered. Two numerical examples with small matrices are included.

Key words. bordered matrix, M-matrix, condition number, Markov chain

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

1. Introduction. Consider the problem of finding the right eigenvector p corresponding to the simple zero eigenvalue of the $n \times n$ matrix A, i.e.,

$$Ap = 0 \tag{1}$$

subjected to some scaling constraint

$$e^T p = 1,$$

for some vector e not deficient in p.

Note that the exact form or size of the vector e does not affect the problem (1) theoretically, as the answer p is only unique up to an arbitrary scaling factor. It is crucial only to the conditioning of the numerical solution process used.

In the calculation of the stationary distribution vector of an ergodic Markov chain, the vector e is usually chosen to be the left eigenvector

$$e_1 = (1, \cdots, 1)^T \tag{2}$$

with A being an M-matrix. This corresponds to the fact that $A = I - Q^T$ for some stochastic matrix Q and any row-sum of Q, being the sum of the probabilities, equals to unity.

We shall assume that A is an M-matrix with e_1 in (2) being a left eigenvector corresponding to the simple zero eigenvalue for the rest of the paper, although we shall discuss the case for a general matrix A in § 7.

The solution vector p can then be proved to be positive. Solutions obtained using scaling schemes with e not chosen to be e_1 in (2) can, of course, be scaled back to satisfy $e_1^T p = \|p\|_1 = 1$ with ease. (The effect on the errors of these solutions will be considered later.)

Some background information on M-matrices and Markov chains can be found in the references listed at the end of this paper (e.g. [3], [29]). Various numerical algorithms and applications were suggested and analysed ([20], [24], [26], [28], [35], [38]). The methods divide roughly into three categories:

(I) Rank-1 update methods—the equation in (1) with the scaling scheme is modified to become a nonsingular system of equations by adding rank-1 terms.

* Received by the editors June 29, 1987; accepted for publication (in revised form) July 25, 1989.

† Mathematics Department, Monash University, Clayton, Victoria 3168, Australia.

(II) Row/column deletion methods—a row and a column in A are deleted to yield a nonsingular system.

(III) Other methods, such as QR decomposition with column pivoting [4].

A detailed comparison of these methods will be a major undertaking, especially for problems with large matrices. More work has to be done in this area and the results will be reported elsewhere.

In this paper, we only consider methods related to the first two categories. The first method, suggested by [21] and [26] and analysed in [1], is singled out for comparison with the new algorithm suggested later in this paper. (See also [28], [35], [38].) This is because the new algorithm was motivated by the stability analysis in [1], where instead of deleting a row and a column of A to form a nonsingular submatrix as in [21] and [26], we add a row and a column to A, i.e., to form a bordered matrix of A, to obtain a nonsingular system (cf. [5], [22], [27]). We suggest that the new algorithm is better conditioned than the old one. It is thus natural to consider the new algorithm through looking at the old one.

Note that the applications of bordered systems to continuation methods for bifurcation problems [6], [7], [30], [34] or corrections to approximate eigensystems [8], [9], [17], [40] amount to inhomogeneous problems, in comparison to problem (1). The corresponding conditioning problem has been investigated in [6], [7], [34].

The conditioning of problem (1) has been examined thoroughly by Wilkinson and others [25], [42] in the general context of algebraic eigenvalue problems. For the special case when A is an M-matrix, analysis have been done through the use of the group inverse $A^{\#}$ [21], [23], [31], [32] or the smallest positive singular value $\sigma_{n-1}^{(A)}$ of A [1], [26].

Finally and without getting into details, algorithms for problem (1) can usually be shown to be closely related to the inverse iteration [36], [40], [42].

The plan for this paper is as follows. In § 2, we introduce the algorithm suggested in [21] and [26] and reiterate some stability results from [1]. In § 3, the new algorithm based on the bordered matrix technique will be presented. We then investigate the conditioning of the problem (1) in § 4, and the stability of the new algorithm in § 5. Two simple numerical examples are included in § 6. In § 7, the general case when A is not an M-matrix is discussed, and the paper is concluded in § 8.

2. Algorithm 1.

The following was suggested by [21] and [26] and analysed in [1].

ALGORITHM 1.

Step 1. Find j and k such that the submatrix B formed by deleting the jth row and kth column of A is invertible; i.e.,

$$A = P_j \begin{bmatrix} B & y \\ z^T & \alpha \end{bmatrix} P_k^T \tag{3}$$

where P_i is the permutation matrix that exchanges the ith and nth rows.

Step 2. Solve by LU or QR decomposition [25] the subsystem

$$B\hat{x} = -y. \tag{4}$$

Step 3. Let

$$x = P_k \begin{bmatrix} \hat{x} \\ 1 \end{bmatrix},$$

and scale x to form p, i.e., $p = \gamma x$, such that $\|p\|_1 = 1$.

In [1], it has been proved that, for some number V_{kn},

$$\frac{\sigma_{n-1}^{(A)}}{(|V_{kn}|^{-1}+1)(\sqrt{n}+1)} \leqq \sigma_{n-1}^{(B)} \leqq \sigma_{n-1}^{(A)}, \tag{5}$$

where $\sigma_i^{(W)}$ denotes the ith singular value of the matrix W.

V_{kn} is the kth component of the nth singular vector V_n of A (corresponding to the zero singular value), and $|V_{kn}|^{-1} \leqq \sqrt{n}$. See [1] for details. Note that V_n is a multiple of p.

As $\sigma_{n-1}^{(A)}$ is considered to be a condition number for problem (1) [26] and $\sigma_{n-1}^{(B)}$ that for (4), it was claimed from (5) that solving problems (1) and (4) involves the same conditioning, and thus solving problem (1) by Algorithm 1 is, in this sense, numerically stable. The lower bound in (5) can be optimized through choosing k in Algorithm 1 to be the index of the maximum component of V_n. It can be done after applying Algorithm 1 with some initial arbitrary value of k to obtain some initial approximation $p^{(0)}$ to p, select k from $p^{(0)}$ (as it is a multiple of V_n), and rerun Algorithm 1 if appropriate.

Such analysis was inadequate and incomplete in the following sense:

(a) The quantity $\sigma_{n-1}^{(A)}$ only was used in reflecting the conditioning of the problem in (1), and the usual condition number

$$s_n \equiv \cos(e_1, p) \equiv \cos\theta_n = \frac{|e_1^T p|}{\|e_1\|_2 \cdot \|p\|_2} \tag{6}$$

by Wilkinson [42] is not explicitly involved. (Here $\cos(e_1, p)$ denotes the cosine of the angle θ_n between the vectors e_1 and p, as in standard references such as [37], [42].) It is more than a simple rescaling of results, as the effect of $\|p\|_2$ on the conditioning is lost, due to a specific scaling scheme for p, and absolute errors were considered instead of relative errors.

(b) Similarly, $\sigma_{n-1}^{(B)}$ was used to represent the conditioning of (4), and not the usual condition number

$$\kappa_2(B) \equiv \|B\|_2 \cdot \|B^{-1}\|_2 = \frac{\sigma_1^{(B)}}{\sigma_{n-1}^{(B)}}, \tag{7}$$

thus ignoring the scaling effect of $\|B\|_2$. (Note that B is $(n-1)\times(n-1)$ and $\sigma_{n-1}^{(B)}$ is its smallest singular value.)

It is easy to show that $\sigma_1^{(A)} \geqq \sigma_1^{(B)}$ so an upper bound of $\kappa_2(B)$ in terms of $\kappa_2(A)$ can be obtained together with (5). However, a lower bound for $\sigma_1^{(B)}$ is needed if a lower bound of $\kappa_2(B)$ in terms of $\kappa_2(A)$ is also required (which will be the case if we are interested in the detection of ill-conditioning of problem (1) when solving (4)). Such a bound for $\sigma_1^{(B)}$ can be obtained, using similar techniques as in this paper, but is not of any interest, because of the problems mentioned here.

(c) The lower bound in (5) can be poor and virtually zero for a large value of n, which indicates possible ill-conditioning of Algorithm 1.

(d) Algorithm 1 and the analysis in [1] work only for the case when A is an M-matrix and $e = e_1$ [as in (2)] for the scaling scheme.

3. Algorithm 2. Based on the analysis in [1] and the classical perturbation analysis for algebraic eigenvalue problems by Wilkinson and others [25], [42], the approach of deleting a row and a column in Algorithm 1 seems to demonstrate some potential weaknesses (see (a)-(d) in the previous section). These weaknesses may be real, or may well be spurious because of ineffective use of available information in the stability analysis. Anyhow, the proof of (5) [1] indicates that the factor $(\sqrt{n}+1)$ can be eliminated by using a bordered system of A instead.

ALGORITHM 2.

Step 1. Scale A (by multiplication of a real constant) so that $\|A\|_2 \approx 1$.

Step 2. Solve the bordered system

$$\tilde{A}\tilde{x} \equiv \begin{bmatrix} A & e \\ e^T & 0 \end{bmatrix} \begin{bmatrix} x \\ \beta \end{bmatrix} = \begin{bmatrix} e \\ 1 \end{bmatrix}, \tag{8}$$

where $e = e_1/\sqrt{n}$ (i.e., $\|e\|_2 = 1$). β should be equal to unity in exact arithmetic, and can be used as a check on the numerical behaviour of the algorithm.

Step 3. Scale x back to form p according to any desirable scaling scheme.

Step 1 is a simplified version of the more elaborated scaling of A to D_1AD_2 using diagonal matrices D_i $(i=1,2)$, so that the singular values of A satisfy the inequalities $\sigma_{n-1}^{(A)} \leqq 1 \leqq \sigma_1^{(A)}$, from the stability analysis in § 5. The more complicated scaling scheme does not seem to be justifiable, unless extra requirements exist. In practice, we want to satisfy $\|A\|_2 \approx 1$ using a scaling factor equal to a power of 2 for optimal computer implementation.

The vectors e in Step 2 in (8) must be nondeficient in the right and left eigenvectors corresponding to the zero eigenvalue, respectively, to ensure the invertibility of $\tilde{A}$ [5]. Note also from the scaling scheme for (1) that the problem will not be well posed if e is deficient in the right eigenvector. The angle between the left and the right eigenvectors features prominently in the stability analysis later.

We shall see that the bordered system in Step 2, together with the scaling vector e chosen to be $e_1/\sqrt{n}$, yield a stability result similar to that in (5) but without the dreaded $\sqrt{n}$ factor:

$$\kappa_2^2(A) \leqq \kappa_2^2(\tilde{A}) \leqq \mathscr{C}_1 \cdot \kappa_2^2(A) + \mathscr{C}_2, \tag{9}$$

where the $\mathscr{C}_i$'s are constants dependent on $s_n \equiv \cos(e, p)$ but otherwise small under normal circumstances, and a condition number of the problem in (1) will be proved to be $\kappa_2(A)$.

For well-conditioned $\tilde{A}$, x can then be accurately obtained so long as the relative error in x is not large and x is not too small relative to β. The scaling scheme $e^Tx = 1$, which implies $\|x\| \geqq 1$, ensures that the latter situation will not occur. The inequalities in (9) together with the scaling scheme thus indicate well-conditioning for Algorithm 2.

4. Conditioning of the problem.

Consider the simple eigenvalue λ_k and its eigenvector x_k of A, i.e.,

$$Ax_k = \lambda_k x_k,$$

and the corresponding perturbed eigenvalue problem, for some perturbation parameter ε and $\|F\|_2 = 1$,

$$(A + \varepsilon F)x_k(\varepsilon) = \lambda_k(\varepsilon)x_k(\varepsilon).$$

Assume that the eigenvectors are scaling to have unit length.

From [25, pp. 202–204] (see also [37], [42]), we have

$$\left|\frac{d\lambda_k(0)}{d\varepsilon}\right| = \frac{|y_k^H F x_k|}{|y_k^H x_k|} \leqq |s_k|^{-1}, \qquad s_k \equiv \cos(x_k, y_k) \tag{10}$$

where y_k is the left eigenvector of A corresponding to λ_k and x_k. We also have the asymptotic series

$$x_k(\varepsilon) = x_k + \varepsilon \sum_{i \neq k} \left\{ \frac{y_i^H F x_k}{(\lambda_k - \lambda_i) y_i^H x_i} \right\} x_i + O(\varepsilon^2). \tag{11}$$

Note for our problem in (1) that the separations $|\lambda_k - \lambda_i|$ in the first-order term in the above series will have a minimum equal to $|\lambda_{n-1}|$ (λ_{n-1} being the smallest nonzero eigenvalue of A in magnitude and $\lambda_n = 0$). It can be proved that the higher derivatives of $\lambda_k(\varepsilon)$ with respect to ε (and thus the higher-order terms in the corresponding Taylor series) will contain the quantity λ_{n-1}. (See [13], [32] for details of derivatives of eigenvalues and eigenvector; see also [10], [12] for other perturbation analysis results for algebraic eigenvalue problems by the author.)

Alternatively in [1] and [26], it has been shown that $\sigma_{n-1}^{(A)}$ is a condition number of the problem in the following sense. If $\tilde{p}$ satisfies the perturbed system

$$A\tilde{p} = r, \qquad e^T \tilde{p} = 1; \tag{12}$$

of (1) with $e = e_1$, we have, for $\delta p \equiv \tilde{p} - p$,

$$\|\delta p\|_2 \leqq \frac{(1+\sqrt{n})}{\sigma_{n-1}^{(A)}} \cdot \|r\|_2. \tag{13}$$

The result is comparable to that in terms of λ_{n-1} in [25], [37], and [42].

By choosing $e = e_1/\sqrt{n}$ as in (8), we can prove the following lemma.

LEMMA 1. *Let A be a singular $n \times n$ matrix of rank $n-1$. If δp satisfies equation* (12), *then*

$$\|\delta p\|_2 \leqq \frac{\kappa_2(A)}{s_n} \cdot \frac{\|r\|_2}{\|A\|_2}, \tag{14}$$

and

$$\frac{\|\delta p\|_2}{\|p\|_2} \leqq \kappa_2(A) \cdot \frac{\|r\|_2}{\|A\|_2}; \tag{15}$$

with $s_n \equiv \cos(e, p)$ $(= \|p\|_2^{-1})$ *and* $\kappa_2(A) = \|A\|_2 \|A^+\|_2 = \sigma_1^{(A)}/\sigma_{n-1}^{(A)}$.

Proof. From (1) and (12), we have

$$A\delta p = r, \qquad e^T \delta p = 0, \tag{16}$$

which in turn implies, for some γ and with A^+ denoting the (1, 2, 3, 4)- or Penrose-generalized-inverse [2], [23] of A,

$$\delta p = A^+ r + \gamma p. \tag{17}$$

Let the singular value decomposition (SVD) of A be

$$A = UDV^T = (U_1, U_n) \begin{bmatrix} \Sigma & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_n^T \end{bmatrix} = U_1 \Sigma V_1^T, \tag{18}$$

where the matrices U and V are orthogonal and the diagonal matrix $\Sigma = \operatorname{diag}\{\sigma_1^{(A)}, \cdots, \sigma_{n-1}^{(A)}\}$.

Note that

$$U_n = e = e_1/\sqrt{n}, \qquad V_n = \|p\|_2^{-1} p; \tag{19}$$

which, together with the scaling scheme $e^Tp=1$, imply

$$p=\frac{V_n}{V_n^TU_n}. \tag{20}$$

Premultiply (17) with e^T, and together with the second equation in (16), we have

$$\gamma=-e^TA^+r.$$

Substituting γ back into (17) and expressing in terms of the SVD in (18) $A^+=V_1\Sigma^{-1}U_1^T$, we arrive at

$$\delta p=V_1\Sigma^{-1}U_1^Tr-U_n^TV_1\Sigma^{-1}U_1^Trp=QV_1\Sigma^{-1}U_1^Tr \tag{21}$$

with $Q\equiv I-V_nU_n^T/U_n^TV_n$.

It is easy to prove that $\|Q\|_2=1/\cos(e,p)$ by considering the eigenvalues of QQ^T.

From (21), the properties of norms, and the facts that $\|\Sigma^{-1}\|_2^{-1}=\sigma_{n-1}^{(A)}$ and $s_n=\|p\|_2^{-1}$, the results in (14) and (15) can be deduced. □

Note that the appearance of s_n in (14) and (15) in Lemma 1 is in agreement with the results by Wilkinson and others [25], [37], [42], like those in (10) and (11). Also, using $\kappa_2(A)$ instead of $\sigma_{n-1}^{(A)}$ reflects the scaling of the problem, as seen from the relative error inequality in (15). Note from (14) that the condition number s_n affects only the absolute error in p.

The proof of Lemma 1 uses a similar setup as in Theorem 2.1 of [1], but follows a quite different line of analysis. Note also that in [1] e was chosen to be e_1 and the solution p in this paper is $\sqrt{n}$ times that in [1]. As a result, Lemma 1 represents a stronger result than that in Theorem 2.1 of [1], in the sense that the absolute or relative error in p has been proved to be roughly n times smaller than in [1], and because of the appearance of s_n. (It appears that the factor n represents an unintentional and conservative estimate of s_n.) In addition, the proof of Lemma 1 above does not require A to be an M-matrix or e be chosen in any specific form, say $e=e_1$, unlike that of Theorem 2.1 in [1].

Finally, denote the group-inverse [2] of A by $A^\#$, $\|A^\#\|_2$ has been shown to be a condition number for our problem [16], [21], [23], [31]. The condition number s_n is proved in [11] to satisfy

$$\|A^\#\|_2\leqq s_n^{-2}\|A^+\|_2=\frac{1}{s_n^2\sigma_{n-1}^{(A)}}.$$

The group inverse is also involved in the derivatives of eigenvalues and eigenvectors of matrices depending on parameters [13], [32].

5. Stability of Algorithm 2. Consider the eigenvalues of the matrix $\tilde{A}^T\tilde{A}$, where

$$\tilde{A}=\begin{bmatrix}A & e\\ e^T & 0\end{bmatrix}=\begin{bmatrix}U_1 & U_n & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\Sigma & 0 & 0\\ 0 & 0 & 1\\ c_1^T & c_2 & 0\end{bmatrix}\begin{bmatrix}V_1^T & 0\\ V_n^T & 0\\ 0 & 1\end{bmatrix} \tag{22}$$

using the SVD in (18), with $c=\left[\begin{smallmatrix}c_1\\ c_2\end{smallmatrix}\right]=V^Te$.

Note that $c^Tc=c_1^Tc_1+c_2^2=1$, with

$$|c_2|=\cos(e,p)=\cos\theta_n=s_n,\qquad \|c_1\|_2=\sin\theta_n. \tag{23}$$

Equation (22) implies that the matrix $\tilde{A}^T\tilde{A}$ is unitarily similar to

$$\begin{bmatrix}\Sigma & c_1 & 0\\ 0 & c_2 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\Sigma & 0 & 0\\ c_1^T & c_2 & 0\\ 0 & 0 & 1\end{bmatrix}=\begin{bmatrix}\Sigma^2+c_1c_1^T & c_2c_1 & 0\\ c_2c_1^T & c_2^2 & 0\\ 0 & 0 & 1\end{bmatrix}. \tag{24}$$

Consider now only the $n \times n$ submatrix W in (24) by deleting the last row and column. It is a rank-1 update

(25) $$W \equiv \begin{bmatrix} \Sigma^2 & 0 \\ 0 & 0 \end{bmatrix} + cc^T.$$

Using the theory of rank-1 updates in [25] and [42], we have the following inequalities:

(26) $$[\sigma_1^{(A)}]^2 \leqq \sigma_1^{(W)} \leqq [\sigma_1^{(A)}]^2 + 1, \quad \sigma_n^{(W)} \leqq [\sigma_{n-1}^{(A)}]^2, \quad \sigma_n^{(W)} \leqq 1.$$

Inequalities (26) and the form of the matrix in (24) imply

(27) $$[\sigma_1^{(\tilde{A})}]^2 = \max\{1, \sigma_1^{(W)}\}, \qquad [\sigma_{n+1}^{(\tilde{A})}]^2 = \min\{1, \sigma_n^{(W)}\} = \sigma_n^{(W)}.$$

We can then prove the following lemma concerning the lower bound of $\sigma_n^{(W)}$.

LEMMA 2.

(28) $$\sigma_n^{(W)} \geqq \frac{\cos^2\theta_n[\sigma_{n-1}^{(A)}]^2}{[\sigma_{n-1}^{(A)}]^2 + (\sin\theta_n + \cos\theta_n)^2} \geqq \frac{\cos^2\theta_n[\sigma_{n-1}^{(A)}]^2}{[\sigma_{n-1}^{(A)}]^2 + 2}.$$

Proof. From the well-known rank-1 update formulae [25], [42]

$$(N + uv^T)^{-1} = N^{-1} - \frac{N^{-1}uv^TN^{-1}}{1 + v^TN^{-1}u},$$

the inverse of W in (25) equal to

$$W^{-1} = \begin{bmatrix} \Sigma^{-2} & -c_2^{-1}\Sigma^{-2}c_1 \\ -c_2^{-1}c_1^T\Sigma^{-2} & c_2^{-2}(1 + c_1^T\Sigma^{-2}c_1) \end{bmatrix}.$$

Consider $x^TW^{-1}x$ for some unit vector x, the properties of norms and (23) imply

$$[\sigma_n^{(W)}]^{-1} \leqq [\sigma_{n-1}^{(A)}]^{-2}(1 + \tan\theta_n)^2 + \sec^2\theta_n,$$

which in turn implies the result in (28), with

$$\max_\theta(\sin\theta + \cos\theta) = \sqrt{2}. \qquad \square$$

We can then prove the main result stated in (9), in a slightly different form.

THEOREM 3.

(29) $$\kappa_2^2(A) \leqq \kappa_2^2(\tilde{A}) \leqq s_n^{-2}[\mathscr{C}_1\kappa_2^2(A) + \mathscr{C}_2],$$

where

(i) $\mathscr{C}_1 = 5$, $\mathscr{C}_2 = 1$ *when* $\sigma_1^{(A)} \geqq 1 \geqq \sigma_{n-1}^{(A)}$.

(ii) $\mathscr{C}_1 = 2$, $\mathscr{C}_2 = 3 + [\sigma_1^{(A)}]^2$, *or* $\mathscr{C}_1 = 4 + [\sigma_{n-1}^{(A)}]^2$, $\mathscr{C}_2 = 1$, *when* $\sigma_1^{(A)} \geqq 1$ *and* $\sigma_{n-1}^{(A)} \geqq 1$.

(iii) $\mathscr{C}_1 = 3 + 2[\sigma_1^{(A)}]^{-2}$, $\mathscr{C}_2 = 1$, *or* $\mathscr{C}_1 = 2$, $\mathscr{C}_2 = 2\{1 + [\sigma_{n-1}^{(A)}]^{-2}\}$, *when* $\sigma_1^{(A)} \leqq 1$ *and* $\sigma_{n-1}^{(A)} \leqq 1$.

Proof. From (26)-(28), we can deduce

$$\kappa_2^2(A) \leqq \kappa_2^2(\tilde{A}) \leqq s_n^{-2} \cdot \Phi,$$

with

$$\Phi = \{2 + 2[\sigma_1^{(A)}]^{-2} + [\sigma_{n-1}^{(A)}]^2\} \cdot \kappa_2^2(A) + 1$$

or

$$\Phi = 2\kappa_2^2(A) + 1 + 2[\sigma_{n-1}^{(A)}]^{-2} + [\sigma_1^{(A)}]^2.$$

By considering the three different possible cases (i)-(iii), the result in (29) follows. $\square$

Some observations from the above theorem and its proof can be made as follows:

(1) The bounds in (29) are quadratic in $\kappa_2^2(A)$ in the sense that the $\mathscr{C}_i$'s are dependent on the singular values $\sigma_1^{(A)}$ and $\sigma_{n-1}^{(A)}$.

Note that s_n appears in the upper bound of $\kappa_2^2(\tilde{A})$ in (29). Note also that $\mathscr{C}_1$ and $\mathscr{C}_2$ are small except in pathological cases when $\sigma_1^{(A)}$ and $\sigma_{n-1}^{(A)}$ are both very large (case (ii)) or very small (case (iii)) in comparison to unity. In such unlikely ill-scaled events, a scaling factor can be used to scale the matrix A back to case (i) so that $\sigma_1^{(A)} \geqq 1 \geqq \sigma_{n-1}^{(A)}$, and thus Step 1 in Algorithm 2.
The scaling factor may be estimated using generalizations of the techniques described in [14] and [15] for the estimation of condition numbers in the infinity norm. Simpler schemes may well be adequate and more work has to be done in this area.

(2) For ill-scaled problems or ill-conditioned problems with small s_n, iterative refinements [40], [42] may have to be used in solving the linear system in (8) in Step 2 of Algorithm 2.

(3) It is easy to see that $\tilde{A}$ is a rank-2 update of the matrix A. The updating techniques for M-matrices [18]-[20], [41] may be used to decompose $\tilde{A}$ into LU factors when solving (8) in Algorithm 2.
This is, in fact, the only application of the M-matrix property of A in Algorithm 2 and the related analysis, totally contrasting Algorithm 1 and its analysis in [1]. We have not made use of the special form of e_1 in (2) either. It is unclear how such additional properties of A can be utilised beneficially.

Finally, similar results as in Lemma 2 and Theorem 3 can be proved by considering the matrix [from (22)]

$$\begin{bmatrix} \Sigma & 0 & 0 \\ 0 & 0 & 1 \\ c_1^T & c_2 & 0 \end{bmatrix},$$

its inverse

$$\begin{bmatrix} \Sigma^{-1} & 0 & 0 \\ -c_2^{-1}c_1^T\Sigma^{-1} & 0 & c_2^{-1} \\ 0 & 1 & 0 \end{bmatrix},$$

and the property of matrix norms $\|N\| = \sup_{\|x\|,\|y\|=1} |y^T N x|$. Upper bounds can then be obtained by other properties of norms and inequalities, and lower bounds from careful selection of x and y. This alternative approach yields worse bounds than those of the results in this section, and so has not been used. However, the techniques can be used for more complicated situations, e.g., when the bordered matrix

$$\tilde{A} \equiv \begin{bmatrix} A & q \\ e^T & \alpha \end{bmatrix}$$

is considered, instead of that in (8) with $\alpha = 0$.

6. Numerical examples. The main aim of this paper concerns the bounds for the condition numbers in (5) and Theorem 3, and it is not intended to compare the numerical algorithms in detail. Indeed, we do not want to claim any superiority of the algorithms suggested here. The examples here are only for the illustrative purposes of showing that Algorithm 2 actually works, and of comparing the bounds for the condition numbers.

Elsewhere we shall consider more detailed numerical comparisons of algorithms suggested in earlier sections (and the generalizations in later sections) and other methods, especially when n is large.

Early results for other test matrices ($n < 100$) are promising, agreeing in the main with the observations made in this section.

We consider two examples defined by the matrices $A = A_i \equiv (I - Q_i)^T$, with Q_1 from [21, p. 10] and Q_2 from [39, p. 283]:

$$Q_1 = 10^{-1}\begin{bmatrix} 7.40 & 1.10 & 0 & 0 & 0 & 0 & 0 & 1.50 \\ 0 & 6.78 & 0 & 0 & 0.11 & 0 & 0 & 3.00 \\ 0 & 0 & 0 & 4.00 & 0 & 0 & 0 & 6.00 \\ 0 & 0 & 0 & 6.69 & 0.11 & 0 & 0 & 3.20 \\ 0 & 0 & 0 & 0 & 9.12 & 0 & 0 & 0.88 \\ 0 & 0 & 0 & 0 & 0 & 7.40 & 0 & 2.60 \\ 0 & 0 & 0 & 0 & 0 & 0 & 8.70 & 1.30 \\ 1.50 & 0 & 0.47 & 0 & 0 & 0.55 & 2.70 & 4.78 \end{bmatrix},$$

$$Q_2 = 10^{-5}\begin{bmatrix} 85000 & 0 & 14900 & 90 & 0 & 5 & 0 & 5 \\ 10000 & 65000 & 24900 & 0 & 90 & 5 & 0 & 5 \\ 10000 & 80000 & 9960 & 30 & 0 & 0 & 10 & 0 \\ 0 & 40 & 0 & 70000 & 29950 & 0 & 10 & 0 \\ 50 & 0 & 40 & 39900 & 60000 & 10 & 0 & 0 \\ 0 & 50 & 0 & 0 & 5 & 60000 & 24990 & 15000 \\ 3 & 0 & 3 & 4 & 0 & 10000 & 80000 & 9990 \\ 0 & 5 & 0 & 0 & 5 & 19990 & 25000 & 55000 \end{bmatrix}.$$

The resulting matrices A_i are M-matrices, although Q_2 is not stochastic (because the sixth row does not sum to unity) and e_1 is not the left eigenvector for A_2. However, Q_2 can be made stochastic by changing the (6, 2) element from 0.00050 to 0.00005 with the corresponding matrix A denoted by A_3. We can consider A_2 to be a perturbation of A_3, with the error size equal to 0.00045. Application of the numerical algorithms to A_2 will demonstrate the effects of perturbation or errors.

The computations were performed on the VAX-16 system in the Monash University Computer Centre, using the well-known numerical linear package MATLAB [33]. Linear equations have been solved using Gaussian elimination with partial pivoting. The machine accuracy is around 10^{-15}.

For A_1 and A_3, the computed p's are accurate up to machine accuracy using Algorithms 1 and 2. The actual results are not interesting and thus will be ignored here. The optimal choice of $k = 7$ has been made in the computation for Algorithm 1, for A_i $(i = 1, 2, 3)$. The behaviour of the bounds of the condition numbers for A_2 and A_3 are similar, and the comparisons of these bounds are summarized in Table 1. Case (i) in Theorem 3 applies for the bounds in (29) for all the examples in this section.

The bounds in (5) and (29) behave as expected, with the bounds in (29) tighter. Note also that the bounds for $\kappa_2(B)$ constructed from (5) will be even less tight, when bounds for $\sigma_1^{(B)}$ are available. As mentioned before, the quantity $\sigma_{n-1}^{(B)}$ is not representative enough for use as a condition number, and the bounds in (5) for both A_1 and A_2

TABLE 1

	A_1	A_2
Upper bound in (5)	0.095	0.00022
$\sigma_{n-1}^{(B)}$	0.067	0.000093
Lower bound in (5)	0.0073	0.000021
$\kappa_2(B)$	19.43	13744.79
Upper bound in (29)	49.8^2	13974.24^2
$\kappa_2^2(\tilde{A})$	16.9^2	5850.61^2
Lower bound in (29)	13.8^2	5707.65^2

are out by roughly one order of magnitude, especially for the lower bounds. The bounds in (29) are comparatively better. The condition number $\kappa_2(B)$ is also larger than $\kappa_2(\tilde{A})$, especially for A_2.

The lower bounds in (29) perform well in terms of tightness, with the upper bounds overestimating by two or three times. It may be the effect of the value of $\mathscr{C}_1 = 5$ in case (i) of Theorem 3.

The computation of p for A_2 is summarized in Table 2. The error in the computed $\tilde{p}$ from Algorithm 1 is larger than that from Algorithm 2. Similar observations can be made for the residuals, which are of the same order as the perturbation from A_3 to A_2, i.e., 0.00045. Recall from Table 1 that the condition number for Algorithm 1 ($\kappa_2(B) = 13744.79$) is larger than that for Algorithm 2 ($\kappa_2(\tilde{A}) = 5850.61$). Note also that the residuals are a lot smaller than the actual errors in $\tilde{p}$.

7. When A is not an M-matrix. The results presented so far in this paper have nothing to do with the fact that A is an M-matrix or the special form of the left eigenvector e_1 as in (2), apart from one particular comment [(3) in § 5, after Theorem 3]. For a general matrix A, Algorithm 2 and the related analysis will still work, if we have prior knowledge of the left eigenvector. Because determining the left eigenvector is adjoint to that of the right eigenvector and because it involves the same degree of difficulty, prior knowledge of the left eigenvector seems, in general, to be too much to ask. We thus suggest the following modified version of Algorithm 2, which requires no prior knowledge of the left eigenvector.

ALGORITHM 3.

Step 1. Scale A (by multiplication of a real constant) so that $\|A\|_2 \approx 1$.

Step 2. Solve the bordered system

$$\tilde{A}\tilde{x} \equiv \begin{bmatrix} A & q \\ e^T & 0 \end{bmatrix} \begin{bmatrix} x \\ \beta \end{bmatrix} = \begin{bmatrix} q \\ 1 \end{bmatrix}, \tag{30}$$

where $e = q$ is a randomly chosen vector of unit length. β should be equal to unity in exact arithmetic, and can be used as a check on the numerical behaviour of the algorithm.

Step 3. Scale x to unit length to form $\hat{x}$.

Step 4. Repeat Step 2 using $e = q = \hat{x}$.

Step 5. Scale x back to p using any desirable scaling scheme.

Heuristically, the vector q must only be nondeficient in the left eigenvector U_n for the Algorithm 2 to work. For a well-conditioned problem, the right eigenvector p should have a large component in the direction of U_n. Hopefully, the initial random vector q in Algorithm 3 is going to have a sufficiently large component in U_n and yield an intermediate answer $\hat{x}$ that is reasonably close to p. Choosing $e = q = \hat{x}$ will then ensure that e and q have large components of V_n and U_n, respectively, and an accurate answer is produced.

TABLE 2

	Algorithm 1	Algorithm 2
$\|\text{Error in computed } \tilde{p}\|_2$	0.16	0.037
$\|A\tilde{p}\|_2$	9.2×10^{-5}	4.2×10^{-5}

To put Algorithm 3 (Steps 2 and 4, in particular) on a firmer basis, the following stability theorem (c.f. Theorem 3) for the bordered system with $e = q \neq U_n$ is proved. (The theorem does not make use of the restriction that $e = q$, so is slightly more general.)

THEOREM 4. *With notation similar to Theorem* 3 *and*

$$\cos\theta_n \equiv \cos(e, V_n), \qquad \cos\phi_n \equiv \cos(q, U_n),$$

we have

$$\kappa_2^2(A) \leqq \kappa_2^2(\tilde{A}) \leqq \frac{2}{\cos\theta_n \cos\phi_n} \cdot [\mathscr{C}_1 \kappa_2^2(A) + \mathscr{C}_2 \kappa_2(A) + \mathscr{C}_3], \tag{31}$$

where

(i) $\mathscr{C}_1 = 4 + \sin\phi_n$, $\mathscr{C}_2 = 1 + 2\sin\phi_n$, $\mathscr{C}_3 = 2$, *when* $\sigma_1^{(A)} \geqq 1 \geqq \sigma_{n-1}^{(A)}$.

(ii) $\mathscr{C}_1 = 1$, $\mathscr{C}_2 = 2\sin\phi_n + \sigma_1^{(A)}$, $\mathscr{C}_3 = 4 + \sigma_1^{(A)}\sin\phi_n + [\sigma_1^{(A)}]^2$, *or* $\mathscr{C}_1 = 2 + \sin\phi_n + \sigma_{n-1}^{(A)} + [\sigma_{n-1}^{(A)}]^2$, $\mathscr{C}_2 = 1 + \sin\phi_n + \sigma_{n-1}^{(A)}\sin\phi_n$, $\mathscr{C}_3 = 2$, *when* $\sigma_1^{(A)} \geqq 1$ *and* $\sigma_{n-1}^{(A)} \geqq 1$.

(iii) $\mathscr{C}_1 = 3 + [\sigma_1^{(A)}]^{-1} + \sin\phi_n[\sigma_1^{(A)}]^{-2}$, $\mathscr{C}_2 = 2\sin\phi_n + [\sigma_1^{(A)}]^{-1}$, $\mathscr{C}_3 = 2$, *or* $\mathscr{C}_1 = 1$, $\mathscr{C}_2 = 1 + \sin\phi_n + \sin\phi_n[\sigma_{n-1}^{(A)}]^{-1}$, $\mathscr{C}_3 = 3 + \sin\phi_n + [\sigma_{n-1}^{(A)}]^{-1} + [\sigma_{n-1}^{(A)}]^{-2}$, *when* $\sigma_1^{(A)} \leqq 1$ *and* $\sigma_{n-1}^{(A)} \leqq 1$.

Proof. Similar to the deduction in § 5, consider the eigenvalues of the matrix $\tilde{A}^T\tilde{A}$, where

$$\tilde{A} = \begin{bmatrix} A & q \\ e^T & 0 \end{bmatrix} = \begin{bmatrix} U_1 & U_n & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Sigma & 0 & d_1 \\ 0 & 0 & d_2 \\ c_1^T & c_2 & 0 \end{bmatrix} \begin{bmatrix} V_1^T & 0 \\ V_n^T & 0 \\ 0 & 1 \end{bmatrix} \tag{32}$$

using the SVD of A in (18), with

$$c = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = V^T e, \qquad d = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} = U^T q.$$

Note that $\|c\|_2 = \|d\|_2 = 1$ and

$$|c_2| = \cos(e, V_n) = \cos\theta_n, \qquad \|c_1\|_2 = \sin\theta_n \tag{33}$$

(with s_n no longer equal to $\cos\theta_n$ but the cosine of the angle between U_n and V_n) and

$$|d_2| = \cos(q, U_n) = \cos\phi_n, \qquad \|d_2\|_2 = \sin\phi_n. \tag{34}$$

Equation (32) implies that the matrix $\tilde{A}^T\tilde{A}$ is unitarily similar to

$$N \equiv \begin{bmatrix} \Sigma & 0 & c_1 \\ 0 & 0 & c_2 \\ d_1^T & d_2 & 0 \end{bmatrix} \begin{bmatrix} \Sigma & 0 & d_1 \\ 0 & 0 & d_2 \\ c_1^T & c_2 & 0 \end{bmatrix} = \begin{bmatrix} \Sigma^2 + c_1 c_1^T & c_2 c_1 & \Sigma d_1 \\ c_2 c_1^T & c_2^2 & 0 \\ d_1^T \Sigma & 0 & 1 \end{bmatrix}. \tag{35}$$

The matrix N in (35) can easily be shown to be similar to the rank-1 updated matrix

$$\tilde{N} \equiv \begin{bmatrix} W & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} c_1 \\ 0 \\ \hline c_2 \end{bmatrix} [c_1^T \; 0 | c_2], \qquad W = \begin{bmatrix} \Sigma^2 & \Sigma d_1 \\ d_1^T \Sigma & 1 \end{bmatrix}. \tag{36}$$

Using properties of norms, we can show that

$$[\sigma_1^{(A)}]^2 \leqq \sigma_1^{(W)} \leqq [\sigma_1^{(A)}]^2 + 1 + 2\sigma_1^{(A)}\sin\phi_n, \quad \sigma_n^{(W)} \leqq [\sigma_{n-1}^{(A)}]^2, \quad \sigma_n^{(W)} \leqq 1. \tag{37}$$

These inequalities, together with the application of the theory of rank-1 updates [25], [42] to the matrix $\tilde{N}$ in (36), prove that

$$[\sigma_1^{(A)}]^2 \leqq [\sigma_1^{(\tilde{A})}]^2 \leqq [\sigma_1^{(A)}]^2 + 2\sigma_1^{(A)}\sin\phi_n + 2, \qquad [\sigma_{n+1}^{(\tilde{A})}]^2 \leqq [\sigma_{n-1}^{(A)}]^2. \tag{38}$$

The lower bound for $\sigma_{n+1}^{(\tilde{A})}$ in terms of $\sigma_{n-1}^{(A)}$ can be obtained from the inverse N^{-1} in the form

$$\begin{bmatrix} \Sigma^{-1}Z\Sigma^{-1} & -c_2^{-1}\Sigma^{-1}Z\Sigma^{-1}c_1 & -\Sigma^{-1}Zd_1 \\ -c_2^{-1}c_1^T\Sigma^{-1}Z\Sigma^{-1} & c_2^{-2}(1+c_1^T\Sigma^{-1}Z\Sigma^{-1}c_1) & c_2^{-1}c_1^T\Sigma^{-1}Zd_1 \\ -d_1^TZ\Sigma^{-1} & c_2^{-1}d_1^TZ\Sigma^{-1}c_1 & 1+d_1^TZd_1 \end{bmatrix},$$

with $Z\equiv(I-d_1d_1^T)^{-1}$.

Standard manipulations using properties of norms, (33) and (34) imply

$$\|Z\|_2 \leqq \frac{1}{\cos^2\phi_n}, \qquad \|c_2^{-1}c_1\|_2 = \tan\theta_n;$$

and in turn the lower bound

$$\begin{aligned}[\sigma_{n+1}^{(\tilde{A})}]^2 &\geqq \frac{[\sigma_{n-1}^{(A)}]^2\cos^2\theta_n\cos^2\phi_n}{\cos^2\phi_n[\sigma_{n-1}^{(A)}]^2+[\sigma_{n-1}^{(A)}(1+\cos\theta_n)+\sin\theta_n]^2} \\ &\geqq \frac{[\sigma_{n-1}^{(A)}]^2\cos^2\theta_n\cos^2\phi_n}{2[\sigma_{n-1}^{(A)}+1]^2},\end{aligned} \tag{39}$$

with standard bounds for the trigonometric functions.

The result in (31) is then proved using the bounds in (38) and (39), and the fact that

$$\sigma_1^{(A)}=\kappa_2(A)\sigma_{n-1}^{(A)}, \qquad [\sigma_{n-1}^{(A)}]^{-1}=\kappa_2(A)[\sigma_1^{(A)}]^{-1}. \qquad \square$$

The bounds are not optimal as the coefficients can be sharper, in an effort to make the bounds reasonably simple. Also, similar and coarser bounds can be obtained using techniques indicated at the end of § 5.

Using the scaling strategy in Step 1 in Algorithm 3, and the bounds

$$1\leqq\kappa_2(A)\leqq\kappa_2^2(A), \qquad \sin\phi_n\leqq 1,$$

(31) implies

$$\kappa_2(A)\leqq\kappa_2(\tilde{A})\leqq\frac{2\sqrt{5}}{\sqrt{\cos\theta_n\cos\phi_n}}\cdot\kappa_2(A),$$

and thus the equivalence of the conditioning of problem (1) and Step 2 or 4 of Algorithm 3.

The factors $\sin\phi_n$ in (31) in Theorem 4 reduce to zero in (29) in Theorem 3. The results in (29) are not exactly the same as that in (31) after substituting zero for $\sin\phi_n$ because of different usage of various inequalities. The factors $\sin\phi_n$ can obviously be replaced by unity, but the relationship between Theorems 3 and 4 is then lost.

The factor $\cos\theta_n\cos\phi_n$ in (31) in Theorem 4 indicates that the equivalence of the theoretical problem in (1) and the problem solved by Steps 2 and 4 in Algorithm 3 is sharp if and only if both θ_n and ϕ_n are small. The numerical algorithm is then well conditioned if and only if e in (30) is not too deficient in V_n (i.e., θ_n small) and q is not too deficient in U_n (i.e. ϕ small).

In choosing $e=q$ and e hopefully not too deficient in V_n in Step 4, the well-conditioning of the problem (i.e., V_n not deficient in U_n) should ensure that q is not too deficient in U_n.

The condition number $s_n\equiv\cos(U_n, V_n)=\cos\omega$ does not feature directly in (31), but is related to θ_n and ϕ_n in the following sense. The angle ω is small if and only if θ_n and ϕ_n are small when we choose $e=q$. In general when $e\neq q$, s_n will then be unrelated to θ_n or ϕ_n, and s_n will then affect only the conditioning of problem (1) in the sense of Lemma 1. However, it is not obvious how else q can be chosen, in order to ensure the well-conditioning of the algorithm.

Theorem 4 can be easily generalized to deal with bordered systems with A having an m-multiple zero eigenvalue, and e and q being $n \times m$ matrices. The results in Theorem 4 still hold, with θ_n and ϕ_n replaced by the maximum angles $\theta_{\max} = \arccos \| V_n^T e \|_2$ and $\phi_{\max} = \arccos \| U_n^T q \|_2$, respectively.

8. Conclusions. A direct algorithm for calculating the null vector of a general matrix A has been proposed, with the implications on the calculation of the stationary distribution vector of an ergodic Markov chain discussed. It has proved that the conditioning of the algorithm (and its generalizations) is comparable to that of the original problem. Wilkinson's condition number s_n for algebraic eigenvalue problems and $\kappa_2(A)$ have been shown to have important roles in the conditioning of the problem and the related algorithms.

We have not considered the problems with structured matrices A, as in [28]. When Gaussian elimination with partial pivoting is used to solve the bordered system in (8) or (30), a dense vector q does not affect the improvement in efficiency because of the structures in A. We may choose the vector e to be structured in some way to improve the efficiency of solving the bordered systems. A balance then must be maintained between efficiency and conditioning, but more work must be done in this area.

Finally, it is strange to see the results in [1] for the special case, when A is an M-matrix with $e = e_1$, appearing to be not as satisfactory as the ones in this paper. In addition, it is not clear how such properties can be utilized in the analysis of this paper. Intuitively, more information should yield better results. It may be in this case that the additional property of A being an M-matrix is either irrelevant or has not been fully exploited. This apparent paradox is still unresolved.

Acknowledgments. The author thanks Dr. Gerald Moore, The Imperial College of Science and Technology, University of London, UK, for several motivating discussions. He also thanks Professor A. K. Cline, Department of Computer Sciences, University of Texas at Austin, Austin, Texas, and an anonymous referee for their many invaluable comments and suggestions.

REFERENCES

[1] J. L. BARLOW, *On the smallest positive singular value of a singular* M-*matrix with applications to ergodic Markov chains*, SIAM J. Algebraic Discrete Methods, 7 (1986), pp. 414-424.
[2] A. BEN-ISRAEL AND T. N. E. GREVILLE, *Generalized Inverses Theory and Applications*, John Wiley, New York, 1974.
[3] A. BERMAN AND R. J. PLEMMONS, *Nonnegative Matrices in the Mathematical Sciences*, Academic Press, New York, 1979.
[4] C. H. BISCHOF, *A parallel* QR *factorization algorithm using local pivoting*, Tech. Report, Department of Computer Science, Cornell University, Ithaca, NY, 1988.
[5] J. W. BLATTNER, *Bordered matrices*, J. Soc. Indust. Appl. Math., 10 (1962), pp. 528-536.
[6] T. F. CHAN, *Techniques for large sparse systems arising from continuation methods*, in Numerical Methods for Bifurcation Problems, T. Küpper, H. D. Mittelmann, and H. Weber, eds., ISNM 7, Birkhäuser Verlag, Basel, 1984.
[7] T. F. CHAN AND D. C. RESASCO, *On the condition of nearly singular matrices under rank-*1 *perturbations*, Linear Algebra Appl., 76 (1986), pp. 223-232.
[8] K.-W. E. CHU AND A. SPENCE, *The improvement of approximate solutions of the integral equation eigenvalue problem*, J. Austral. Math. Soc. Ser. B, 8 (1981), pp. 474-487.
[9] K.-W. E. CHU, *Deferred correction for the ordinary equation eigenvalue problem*, Bull. Austral. Math. Soc., 26 (1982), pp. 445-454.
[10] ———, *Generalizations of the Bauer-Fike theorem*, Numer. Math., 49 (1986), pp. 685-691.

[11] K.-w. E. Chu, *Bordered matrices, singular systems and ergodic Markov chains*, Tech. Report NA/14/86, Mathematics Department, University of Reading, Reading, UK, 1986.

[12] ———, *Exclusion theorems and the perturbation analysis of the generalized eigenvalue problem*, SIAM J. Numer. Anal., 24 (1987), pp. 1114-1125.

[13] ———, *On multiple eigenvalues of matrices depending on several parameters*, SIAM J. Numer. Anal., 27 (1990), to appear.

[14] A. K. Cline, A. R. Conn, and C. Van Loan, *Generalizing the* LINPACK *condition number*, in Numerical Analysis, J. P. Hennart, ed., Lecture Notes in Mathematics #909, Springer-Verlag, New York, 1982.

[15] A. K. Cline, C. B. Moler, G. W. Stewart, and J. H. Wilkinson, *An estimate for the condition number of a matrix*, SIAM J. Numer. Anal., 16 (1979), pp. 368-375.

[16] R. E. Cline, *Inverses of rank invariant powers of a matrix*, SIAM J. Numer. Anal., 5 (1968), pp. 182-197.

[17] L. Collatz, *Functional Analysis and Numerical Mathematics*, Academic Press, New York, 1966.

[18] R. E. Funderlic and R. J. Plemmons, LU *decomposition of* M*-matrices by elimination without pivoting*, Linear Algebra Appl., 41 (1981), pp. 99-110.

[19] R. E. Funderlic, M. Neumann, and R. J. Plemmons, LU *decomposition of generalized diagonally dominant matrices*, Numer. Math., 40 (1982), pp. 57-69.

[20] R. E. Funderlic and R. J. Plemmons, *Updating* LU *factorizations for computing stationary distributions*, SIAM J. Algebraic Discrete Methods, 7 (1986), pp. 30-42.

[21] R. E. Funderlic and C. D. Meyer, jr., *Sensitivity of the stationary distribution vector for any ergodic Markov chain*, Tech. Report ORNL-6098, Engineering, Physics and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN, 1984.

[22] D. Goldfarb, *Modification methods for inverting matrices and solving systems of linear algebraic equations*, Math. Comp., 26 (1972), pp. 829-852.

[23] G. H. Golub and C. D. Meyer, jr., *Using the* QR *factorization and group inversion to compute, differentiate, and estimate the sensitivity probability of Markov chains*, SIAM J. Algebraic Discrete Methods, 7 (1986), pp. 273-281.

[24] G. H. Golub and E. Seneta, *Computation of the stationary distribution of an infinite Markov chain*, Bull. Austral. Math. Soc., 8 (1973), pp. 333-341.

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

[26] W. J. Harrod and R. J. Plemmons, *Comparison of some direct methods for computing stationary distributions of Markov chains*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 453-469.

[27] M. G. Hestenes, *Inversion of matrices by biorthogonalization and related results*, J. Soc. Indust. Appl. Math., 6 (1958), pp. 51-90.

[28] L. Kaufman, *Matrix methods for queuing problems*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 525-552.

[29] J. G. Kemeny and A. L. Snell, *Finite Markov Chains*, Van Nostrand, Princeton, NJ, 1960.

[30] H. B. Keller, *The bordering algorithm and path following near singular points of higher nullity*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 573-582.

[31] C. D. Meyer, jr., *The condition of a finite Markov chain and perturbation bounds of the limiting probabilities*, SIAM J. Algebraic Discrete Methods, 1 (1980), pp. 273-283.

[32] C. D. Meyer and G. W. Stewart, *Derivatives and perturbations of eigenvectors*, SIAM J. Numer. Anal., 25 (1988), pp. 679-691.

[33] C. B. Moler, MATLAB *User's Guide*, Tech. Report CS81-1, Department of Computer Science, University of New Mexico, Albuquerque, NM, 1980.

[34] G. Moore, *Some remarks on the deflated block elimination method*, in Bifurcation: Analysis, Algorithms, Applications, Küpper, Seydel, Tröger, eds., ISNM 79, Birkhaüser Verlag, Basel, 1987.

[35] C. C. Paige, P. H. Styan, and P. G. Wachter, *Computation of the stationary distribution of a Markov chain*, J. Statist. Comput. Simulation, 4 (1975), pp. 173-186.

[36] G. Peter and J. H. Wilkinson, *Inverse iteration, ill-conditioned equations and Newton's method*, SIAM Rev., 21 (1979), pp. 339-360.

[37] G. W. Stewart, *Introduction to Matrix Computations*, Academic Press, New York, 1973.

[38] ———, *A comparison of numerical techniques in Markov modelling*, Comm. ACM, 21 (1978), pp. 144-151.

[39] ———, *Computable error bounds for aggregated Markov chains*, J. Assoc. Comput. Mach., 30 (1983), pp. 271-285.

[40] H. J. Symm and J. H. Wilkinson, *Realistic error bounds for a simple eigenvalue and its associate eigenvector*, Numer. Math., 35 (1980), pp. 133-126.

[41] R. S. Varga and D. Cai, *On the* LU *factorization of* M*-matrices*, Numer. Math., 38 (1981), pp. 179-192.

[42] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Oxford University Press, Oxford, UK, 1965.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 4, pp. 702-712, July 1990

RANDOM SEARCH IN THE PRESENCE OF NOISE, WITH APPLICATION TO MACHINE LEARNING*

S. YAKOWITZ† AND E. LUGOSI‡

Abstract. A search for the global minimum of a function is proposed; the search is on the basis of sequential noisy measurements. Because no unimodality assumptions are made, stochastic approximation and other well-known methods are not directly applicable. The search plan is shown to be convergent in probability to a set of minimizers. This study was motivated by investigations into machine learning. This setting is explained, and the methodology is applied to create an adaptively improving strategy for 8-puzzle problems.

Key words. machine learning, random search, optimization

AMS(MOS) subject classifications. 82L20, 68A35

1. Introduction.

The object of this study is the following problem: A measurable real-valued function $f(x)$ is defined on a domain D in R^d. At each decision time $n=1, 2, \cdots$, the decision maker selects a point, $X(n)$, and observes the number

$$Y(n)=f(X(n))+W(x(n)) \tag{1.1}$$

where $W(X(n))$ is a random variable whose distribution function may depend on the point selected. It is presumed that $W(\ )$ has a constant mean (we take it to be 0) and a uniformly bounded absolute pth moment, $p>2$. Moreover, the variables of the sequence

$$\{W(X(n)): n=1,2,\cdots\}$$

are assumed independent, given the $X(n)$'s, even if some of the x-values are repeated. The goal is to find a sequential search plan that assures that if $f(\ )$ has an essential infimum, say $f_{\min}$, then for any $e>0$,

$$P[f(X(n))>f_{\min}+e]\to 0 \text{ as } n\to\infty. \tag{1.2}$$

We will propose a plan that achieves this consistency property.

The problem was motivated by our investigations of machine learning for solving the 8-puzzle, a puzzle oft-mentioned in the computer science literature (e.g., Doran [7], Doran and Michie [6], Michie and Ross [17], Barr and Feigenbaum [1], Levy [13], and Nilsson [18]). Our idea in machine learning was to parameterize a space of plausible (heuristic) value functions and subgraph searches, so that x, in this context, plays the role of parameter value. In our application, x serves as a weighting factor to balance criteria in a value function and to limit the depth of search. (This artificial intelligence terminology will be defined in § 4. Intuitively speaking, in checkers, the parameter could quantify trade-off between control of the center of the board, protection of pieces, etc.) The initial 8-puzzle configuration is chosen at random, and $f(x)$ is the expected computational effort required, under the decision strategy determined by x,

* Received by the editors December 22, 1986; accepted for publication (in revised form) June 15, 1989. This work was supported in part by National Science Foundation grant DPP-82-19439.

† Systems and Industrial Engineering Department, University of Arizona, Tucson, Arizona 85721.
† Mathematical and Physical Journal for High School Students, Budapest, Martirok u. 24, Hungary.

to put the puzzle into the target position in which the numbers read in ascending order. In our study, computer effort is measured by the number of nodes expanded in reordering the tiles. A search that looks deeply into the game graph usually makes a better move, but it expands more nodes. After having the computer solve a single puzzle with a randomly chosen initial configuration, one gets, effectively, a random observation $Y(x)$ = number of node expansions for a certain initial configuration and for the decision strategy associated with parameter x. The expected value of this observation (over the randomization on initial boards) is $f(x)$. Our claim is that this setting is really quite general and could apply to checkers and chess. Indeed, our confidence in this approach has been fortified by experimental studies with go-moku [22], and an assembly-line balancing problem [24]. Our computational experience with 8-puzzle learning will be reviewed in § 4.

In the systems literature of ten to twenty years ago, a number of random search methods were proposed, most of them being dedicated to finding global minima on the basis of noiseless measurements. The review [9] gives a good summary of the engineering literature, and reference [21] offers more theoretical references. Some algorithms for the noisy case (1.1) have also been posed ([4], [5], [9], [16], [21, § 4]) and in some cases, were shown to be consistent; a strategy in [21] even achieved a convergence rate. Yet for one reason or another, these techniques have not gained popularity. The present approach stems from [21], but we think it to be much more practical and efficient.

Stochastic approximation, simulated annealing, and bandit methods are also related to our stochastic minimization problem. The Kiefer-Wolfowitz [10] strain of stochastic approximation allows for noisy measurements, but unimodality is a fundamental hypothesis to this approach.

Simulated annealing presumes a deterministically observed underlying function. However, it is to be noted that recently Kushner [11] has used simulated annealing ideas to extend stochastic approximation to global searches. One idea from the simulated annealing literature that we have adopted is that of putting a Gibb's distribution on the sample points in the search to be described in the next section. This tactic, which is not absolutely essential, was inspired, in part, by ideas in [8]. Having studied our construct, the reader will be able to devise consistent alternatives.

Traditionally, bandit problems (e.g., Berry and Fristedt [2]) presume that there are only finitely many search parameters. However, this literature is relevant (especially the optimal strategy of Lai and Robbins [12]). Yakowitz and Lowe [23] have explored generalizations to nonparametric, infinite bandit populations that are also aimed at our stochastic minimization model.

Let it be understood at the outset that the search problem having been posed, it can be solved by fairly elementary considerations (although to our knowledge, such solutions have not worked their way into practice). For example, it is known (e.g., Mack and Silverman [15], Devroye [4], and Schuster and Yakowitz [20]) that under fairly general circumstances, nonparametric regression estimators converge uniformly, on bounded domains, to the sampled "target" function. To adapt these developments to the search problem, one can devise a plan that calls for asymptotically sparse sampling to "learn" the underlying function $f(x)$, and at intervening times, sampling at the point that minimizes the inferred regression function, to achieve asymptotically optimal performance.

While such devices can be shown (e.g., Devroye [4]) to possess property (1.2), they seem to us *ad hoc* and wasteful. For instance, there does not seem to be much point to "learning" the objective function accurately in regions that are suboptimal.

The situation, as we see it, is that no compelling methodology for stochastic minimization of a multimodal function has yet emerged, and therefore sensible avenues should be explored. The plan to be revealed next evolved from pragmatic considerations and substantial computational explorations.

2. A search methodology.

2.1. The algorithm.

Components:

1. We presume that the absolute pth moment, $p>2$, of $W(x)$ is bounded, uniformly in x.
2. Let b denote a number greater $2/(p-2)$ and 1. For $i=1, 2, \cdots$, let $N(i)$ be an integer chosen at random from the interval $((i-1)^b, i^b]$. Thus $N(1)=1$. We denote the members of $\{N(i)\}$ as "search times." (At search times new sample points are tested.)
3. Let $p(x)$ be a probability density function whose support is the entire domain D of $f(x)$. (During search times, new values of x will be chosen at random, according to $p(x)$.)
4. $\{T(n): n \geqq 1\}$ is a sequence of positive numbers converging to 0.

Initialization: $n=1$, $\mathrm{NP}=0$, $m_i(0)=0$ for $i=1, 2, \cdots$.

The Procedure: If $n \in \{N(i)\}$, sample and update. $\{n$ is a "sample time"$\}$

1. Set $\mathrm{NP}=\mathrm{NP}+1$, $N_{\mathrm{NP}}=1$. {NP counts the number of sample points}
2. Choose the sample point $X(n)=t(\mathrm{NP})$ at random from D, according to the *pdf* $p(x)$. $\{t(j)$ denotes the jth sample point$\}$
3. Observe a noisy function value:

$$Y(X(n))=f(X(n))+W(X(n)) \tag{2.1}$$

and define

$m_{\mathrm{NP}}(n)=Y(X(n))$. $\{m_j$ denotes the average performance at sample point $t(j)\}$

4. Update resample probabilities: for $i=1, \cdots, \mathrm{NP}$, define the "resample" probabilities by

$$P(i)=\exp(-m_i(n-1)/T(n))/\mathrm{Denom} \tag{2.2}$$

where

$$\mathrm{Denom}=\sum_{j=1}^{\mathrm{NP}} \exp(-m_j(n-1)/T(n)). \tag{2.3}$$

{A point whose average performance is good (low m_i) has the better chance of being picked. As n increases, $T(n)$ decreases, and the chance of good points are even higher. But by our minimal sample schedule in step 7 below, even bad points are picked infinitely often, but relatively seldom. This prevents an optimal point from being overlooked due to a run of "bad luck."}

Go to 7.

Else, if $n \notin \{N(i))$, resample. $\{n$ is not a "sample time"$\}$

5. Choose index I at random according to the probability masses $\{P(i)\}$ in step 4, and make an observation at resample point $X(n)=t(I)$. Let

$$Y(X(n)) := f(X(n))+W(X(n)),$$

and

$N_I := N_I(n)+1\{N_I(n)$ counts the number of samples that have been taken at $t(i)\}$

{*Remark.* Please bear in mind that $N(i)$ is the ith sample time, whereas $N_i(n)$ is the number of observations made at the point $t(i)$ by time n.}

6. Update the mean value at index I according to

$$m_I(n) = ((N_I - 1)/N_I)m_I(n-1) + Y(X(n))/N_i, \tag{2.4a}$$

and set

$$m_i(n) := m_i(n-1), \quad \text{for } i \neq I. \tag{2.4b}$$

7. If $n \in \{N(i)\}$, assure that all points have been sampled at least Min (n) times, where

$$\text{Min}\,(n) = C'[n^{1/b}\log(n)]^{1/(p-2)}. \tag{2.5}$$

(C' is an arbitrary positive constant.) For each $i \leqq \text{NP}$, perform steps (2.4a) and (2.4b) while

$$N_i(n) < \text{Min}\,(n). \tag{2.6}$$

Increment n and $N_i(n)$ at each pass.

8. Set $n = n+1$, and begin the procedure again.

2.2. Experimental example. We have coded this search technique and applied it to several multimodal functions. For example, we took $D = [0, 1]$ and $f(x) = \sin(10\pi x)$, with $W(n)$ being standard normal, and the sample *pdf* $p(\ )$ being uniform on D. To impress our readers (or at least ourselves) with how erratic the noisy samples of this oscillatory function are, we have made a plot (Fig. 1) showing the target function and 100 pairs $(Z(i), Y(i))$, where the $Z(i)$'s are chosen uniformly on D. In Table 1, we have listed averages, over 100-point blocks, of $f(X(i))$, where the search was conducted according to the preceding plan. We set $b = 2$, $p = 10$, and $C' = 1$.

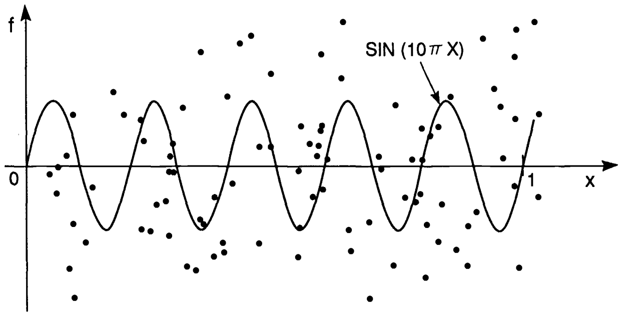


FIG. 1. *The function $f(x)$ and 100 noisy sample pairs.*

The value $f_{\min}$ for this function is, of course, -1.0. The search gets off to a reasonably good start, but many observations are needed to get very close to this optimal value. The reason is that the measure of points in $[0, 1]$ for which $\sin(10\pi x) < -0.95$, say, is about 10 percent, and yet this gives the probability of a randonly chosen test point $t(i)$ satisfying that condition. Keep in mind that the total number of sample points grows only as the square root of the number of observations.

Some improvement can be made by updating the selected value of $t(i)$ by a "Kiefer-Wolfowitz" (KW) step at each resample time. By this plan, we simultaneously search all valleys, and thereby with this modification, under suitable smoothness conditions, assure that the minimum is approached at the fast stochastic approximation

TABLE 1

Average performances during successive blocks of 100 *samples.*

Block number n	Average value over nth block $(1/100) \sum_{i=100n+1}^{100(n+1)} f(X(i))$
0	−0.49
1	−0.78
2	−0.72
3	−0.89
4	−0.83
5	−0.79
6	−0.82
7	−0.84
8	−0.82
9	−0.83

rate, even though the function is not unimodal. We do not dwell on this point because in artificial intelligence applications, such as the 8-puzzle, the objective function is typically discontinuous and the KW idea is inapplicable. In Table 2, we relate the results of redoing the experiment with KW steps as just described.

3. Convergence analysis of the search. Here we show that if the target function $f(x)$ is continuous and has a finite infimum, call it $f_{\min}$ (or even if $f(x)$ is discontinuous, with a finite Lebesgue-essential infimum), then the performance under the search specified in the preceding section converges in probability to the optimal, in the sense that,

$$f(X(n)) \to f_{\min}, \text{ in probability, as } n \to \infty.$$

Let NP = NP (n) continue to denote the number of distinct randomly selected domain points obtained by time n, and $\{t(j),\ 1 \leqq j \leqq \mathrm{NP}\}$, the actual values of these points. Recall that the sequence $\{N(j)\}$ denotes the sample times.

TABLE 2

Average performance during successive blocks of 100 *samples wtih KW steps.*

Block number n	Average value over nth block $(1/100) \sum_{i=100n+1}^{100(n+1)} f(X(i))$
0	−0.55
1	−0.77
2	−0.85
3	−0.81
4	−0.90
5	−0.94
6	−0.87
7	−0.87
8	−0.86
9	−0.87

LEMMA 1. *Let e be a positive number and $f_{\min}$ denote the (essential) infimum of $f(x)$. At resample times n, as $n \to \infty$,*

$$P[X(n) = t(i)\colon m_i(n) \leqq f_{\min} + e] \to 1. \tag{3.1}$$

Proof. Define the set

$$S(n) = \{t(i)\colon i \leqq \mathrm{NP},\ m_i(n) < f_{\min} + e/4,\ i \leqq \mathrm{NP}(n)\}.$$

Then for β sufficiently small but positive, and $\#S$ denoting the number of points in S, as $n \to \infty$,

$$P[\#S(n) > \beta\ \mathrm{NP}(n)] \to 1. \tag{3.2}$$

To confirm this, observe that for any fixed value $t(i)$,

$$P[m_i(n) < f(t(i)) + e/4, \text{all } n = 1, 2, \cdots] > 0.$$

This assertion is a restatement of [3, Lem. 2.8]. Let $Q(t)$ denote the probability above when $t(i) = t$, and set

$$A = \{t\colon m(t) \leqq f_{\min} + e/4\}.$$

Then since $Q(t)$ is positive on A, and since A has positive probability under $p(\ )$, we have that for each sample point $t(i)$,

$$P[m_i(n) \leqq f_{\min} + e] \geqq \int_A Q(t)p(t)\,dt > 0. \tag{3.3}$$

For β not exceeding the above integral, (3.2) is a consequence of the law of large numbers.

Omit the argument n in $S(n)$, $T(n)$, etc., for now. Under the event in (3.2)

$$P[X(n) = t(i)\colon m_i < f_{\min} + e]$$
$$\geqq \frac{\sum_{k \in S} \exp(-m_k/T)}{\sum_{1 \leqq k \leqq \mathrm{Np}} \exp(-m_k/T)}.$$

Let "Num" denote the numerator above. This fraction is bounded from below by

$$\frac{\mathrm{Num}}{[\mathrm{Num} + \mathrm{NP}\exp(-((f_{\min} + e)/T))]}.$$

Then note that if the event in (3.2) holds,

$$\mathrm{Num} \geqq \beta\ \mathrm{NP}\exp(-(f_{\min} + e/2)/T).$$

One concludes from these two relations that the second term in brackets of the denominator of the preceding fraction is asymptotically negligible, as $T \to 0$, and so almost surely, (3.1) is true. □

LEMMA 2. *Let $\{Z(i)\}$ be a sequence of independent 0-mean random variables with finite absolute pth moments, $p > 2$. Then for $S(n) = \sum Z(i)\colon 1 \leqq i \leqq n$, and e a positive number,*

$$P\left[\max_{j \geqq n} |S(j)|/j > e\right] < C/n^{p-2}. \tag{3.4}$$

Proof. As noted in [4, Lem. 2], we have that for some constant C_1,

$$P[|S(j)|/j > e] < C_1/j^{p-1}.$$

Now just sum these terms from n to ∞, and note that the probability of unions of a countable number of events does not exceed the sum of the individual event probabilities. □

THEOREM. *If for some $p>2$ the* pth *absolute moment of $W(t)$ is uniformly (in t) bounded, then as $n\to\infty$,*

$$f(X(n))\to f_{\min},\ \textit{in probability.}$$

Proof. By choice of b, the probability that a given number n is either a sample time or "step 7" time tends to zero as $n\to\infty$. So it suffices to prove the assertion for resample times. Let n be a resample time and $S(n)$ be as in the Proof of Lemma 1. Define the set $B(n)$ (B for "bad") by

$$B(n)=\{t(i)\colon i\leqq \mathrm{NP}, f(t(i))>f_{\min}+e\}.$$

we show that

$$P[X(n)\in B(n)]\to 0.$$

It is apparent that the set $S(n)$ is random. Let $Q(n)$ denote that event $B(n)\cap S(n)$ is not empty. Our plan is to show that $P[Q(n)]\to 0$. Since the content of Lemma 1 is that

$$P[X(n)\in S(n)]\to 1,$$

the theorem follows. We let $Z=Z(n):=B(n)\cap S(n)$. Then

$$P[Q(n)]\leqq \Sigma_Z P[|m_i(n)-f(t(i))|>e]$$
$$\leqq C\Sigma_Z E(1/N_i(n)^{p-2}),$$

where C is the bound of Lemma 2, as applied to $W(x)$. Now by the construct (2.5),

$$N_i(n)^{p-2}\geqq C'n^{1/b}\ln(n),$$

whence

$$P[Q(n)]<C\ \mathrm{NP}/(C'n^{1/b}\ln(n))=O(1/\ln(n)),$$

where we have used that $\mathrm{NP}(n)$ grows as $n^{1/b}$.

In summary,

$$P[X(n)\in B]\leqq P[X(n)\in S(n)\cap B(n)]+P[X(n)\notin S(n)].$$

The first probability on the right vanishes because the probability that the event $S(n)\cap B(n)$ is not the empty set converges to zero, and the second converges to 0, in view of Lemma 1. □

4. An application of random search to machine learning.

In our view, the common artificial intelligence methods for games are determined by three objects:

(i) The game graph,
(ii) An algorithm for machine expansion of a subgraph about any given node, and
(iii) A heuristic value function defined on nodes.

The game graph determines the available decisions or moves from a given state or board position, each allowable action being represented by a branch emanating from the node representing the state. We will display a subgraph later when details of the 8 puzzle are described.

The customary plan for choosing a branch at a given node involves expanding the subtree about that node, and using the value function to assign numbers to the leaves (i.e., the terminal nodes). Then dynamic programming may be used to discern the best decision at the expansion node, with respect to the subgraph and leaf values. This process is repeated at each decision time.

The application of the stochastic minimization scheme related in the preceding section is predicated on the hypothesis that a sequence of similar decision problems presents itself. In the 8-puzzle problem, that sequence will be taken to be puzzles with randomly chosen initial boards. In a sequence of checker games, for example, one can imagine that the randomness arises by variations of an opponent's strategy. We identify the minimization variable x with the vector (x_1, x_2, x_3), where x_1 parameterizes a space of value functions defined on boards B, and x_2, and x_3 similarly index a collection of graph expansion rules. The objective function $f(x)$ is viewed in the machine learning setting as the expected risk of a decision problem, given the decision strategy determined by the value-function/graph-search parameter $x=(x_1, x_2, x_3)$.

Our experience at this writing includes the 8-puzzle, tic-tac-toe, go-moku, a stochastic travelling salesman problem, and assembly-line tuning. The intention of this section is to report our experience in applying our random search scheme to the 8-puzzle. Following Schofield [19], in the starting and target configuration the center square is empty (Fig. 2 and Fig. 3). The tiles are numbered 1 through 9, but without a 5.

2	1	3
4		6
7	9	8

FIG. 2. *A typical starting configuration.*

1	2	3
4		6
7	8	9

FIG. 3. *The target configuration.*

The objective is to rearrange the board by successively moving tiles to the blank space until the arrangement in Fig. 3 is achieved. The measure of performance of a particular puzzle solution is determined by the total number of nodes expanded during the process of reordering the tiles. A search that does not expand very deeply at each board tends to move fast, but the moves are less fruitful than one that expands to deeper levels. The motivation is that the number of nodes expanded ought to be a fairly good indicator of CPU time.

Thus, in connection with the model (1.1), $Y(n)$ is the number of node expansions required to rearrange the nth puzzle, and $f(x)$ is the expected number of search expansions, under value-function and search-parameters (x_1, x_2, x_3). The expectation is over randomly chosen initial boards. There are 20,160 possible initial boards.

In our experiments, at each "game," an initial board was chosen at random from the class of all solvable boards. The subgraph at each node was the full graph about that node up to a given depth x_3, with the proviso that some improvement must be obtained in the value function by level x_2. In Fig. 4 we have expanded the graph, about the board node shown in Fig. 2, to a depth $x_3=2$.

The parametrized value functions were defined on board arrangements B by

$$V(b; x_1)=x_1^*(\mathrm{Vert}\,(B)+\mathrm{Horiz}\,(B))+\mathrm{Manhat}\,(B). \tag{4.1}$$

Here Horiz (B) is the number of order inversions when the board numbers of B are read horizontally, i.e., row by row. Thus in the case of the root board shown in Fig. 2, reading horizontally, we have

2 1 3 4 6 7 9 8

and we observe that 21 and 98 are inverted with respect to the sequence 1, 2, 3, 4, 6, 7, 8, 9. Consequently, for the board shown, Horiz $(B)=2$.

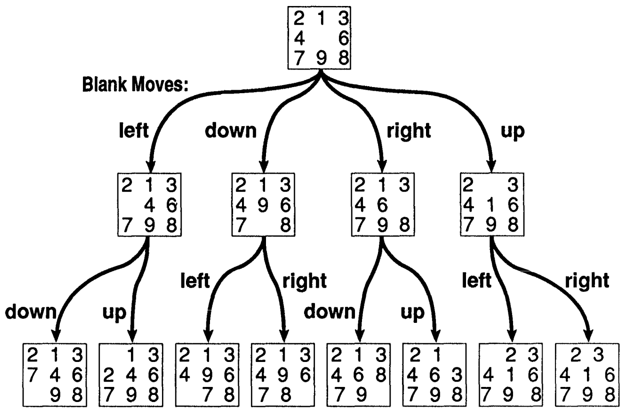


FIG. 4. *An 8-puzzle subgraph.*

Vert (B) is the number of inversions when the tiles are read vertically, column by column, where inversions are in comparison to the ordering 1, 4, 7, 2, 8, 3, 6, 9. In case of the board above, reading vertically, we have

$$2\ 4\ 7\ 1\ 9\ 3\ 6\ 8.$$

Regarding the first four numbers, 2471, we get two vertical counts because 2 improperly precedes 4 and 7. Then we get three more, because 2, 4, and 7 have order inverted with respect to 1. Thus this subsequence has five inversions. The remaining four numbers have a similar pattern of inversion with respect to their proper ordering. Hence Vert $(B) = 10$.

The measure Manhat (for "Manhattan distance") is the sum

$$\text{Manhat}\,(B) = \Sigma d(i)$$

where $d(i)$ is defined to be the minimal number of moves required to get the tile labeled "i" from its current board position to its correct place on a properly arranged board, these moves presumed made in absence of all other tiles. Thus in the board of Fig. 2 Manhat $(B) = 4$, because the tiles 1 and 2 are each off their proper places by one move, as are 8 and 9. The measure Manhat had been previously used by Levy [13].

The law $p(\,)$, for choosing the sample points $t(i)$ is composed on three segments. The variable x_1, indexes the value functions, as in (4.1), and (x_2, x_3) determines the tree expansion as explained above. The law of x_1 is the uniform density on $[0, 3]$. The maximum search depth x_3 is random on the integers 1 to 5, and x_2, the depth by which an improvement must occur, is chosen from integers 1 to, but not including x_3, unless $x_3 = 1$. (From Schofield [19] it is known that the depth of a full graph can be as large as 30, and the average depth is over 20; computationally speaking, the 8-puzzle is not trivial!)

The program organization entailed making the 8-puzzle strategy a procedure, with passing input parameters (x_1, x_2, x_3) as just described. This procedure relays an output Y to the driver program, the value Y being the count of the total number of nodes expanded during puzzle rearrangement. The driver program itself is essentially the same as that used for the sine function experiment discussed in § 2, with obvious accommodations being made for the change in the optimization variable x.

The results of a learning session are summarized in Table 3. The situation is that the expected number of node expansions varies sharply with graph expansion regime. For some parameters causing very shallow searches, the puzzle is never rearranged.

TABLE 3

Average number of node expansions per puzzle.

N	Average number of node expansions in first N puzzles
10	80,720
20	45,010
50	30,210
100	19,730
300	12,920
700	10,190
1000	8,860

(The program terminates the search after 250 tile moves. If the program failed to terminate, the large penalty 11,400 was assigned.)

The point we wish to make is that in this application, marked on-line improvement is attainable while statistical inference is undertaken.

Acknowledgment. The first author is grateful for related discussions with Professor C. Newman.

REFERENCES

[1] A. BARR AND E. A. FEIGENBAUM, *The Handbook of Artificial Intelligence, Vol.* I, William Kaufman, Inc., Los Altos, CA, 1982.

[2] D. A. BERRY AND B. FRISTEDT, *Bandit Problems*, Chapman and Hall, London, 1985.

[3] Y. S. CHOW, H. ROBBINS, AND D. SIEGMUND, *Great Expectations*, Houghton Mifflin Co., Boston MA, 1971.

[4] L. P. DEVROYE, *The uniform convergence of nearest neighbor regression function estimators and their application in optimization*, IEEE Trans. Inform. Theory, 24 (1978), pp. 142-151.

[5] ———, *Progressive global random search of continuous functions*, Math. Programming, 15 (1978), pp. 330-342.

[6] J. E. DORAN AND D. MICHIE, *Experiments with the graph tranverser program*, Proc. Roy. Soc. London Ser. A, 294 (1966), pp. 235-259.

[7] J. E. DORAN, *An approach to automatic problem-solving*, in Machine Intelligence 1, N. L. Collins, and D. Michie, eds., Oliver and Boyd 1967, pp. 105-123.

[8] S. GEMAN AND D. GEMAN, *Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images*, in IEEE Trans. and Pattern Analysis and Machine Intelligence, PAMI-6, 1984, pp. 721-741.

[9] R. A. JARVIS, *Optimization strategies in adaptive control: A selective survey*, IEEE Trans. Systems, Man Cybernet., SMC-5 (1975), pp. 83-94.

[10] J. KIEFER AND J. WOLFOWITZ, *Stochastic estimation of the maximum of a regression function*, Ann. Math. Stat., 23 (1952), pp. 462-466.

[11] H. L. KUSHNER, *Asymptotic global behavior for stochastic approximation and diffusion with slowly decreasing noise effects: Global minimization via Monte Carlo*, SIAM J. Appl. Math., 43 (1987), pp. 169-185.

[12] T. L. LAI AND H. ROBBINS, *Asymptotically efficient adaptive allocation rules*, Adv. in Appl. Math., 6 (1985), pp. 4-22.

[13] D. LEVY, *Computer Gamesmanship*, Simon & Schuster Inc., New York, 1983.

[14] M. LOEVE, *Probability Theory*, Van Nostrand, Princeton, NJ, 1955, p. 238.

[15] Y. P. MACK AND B. W. SILVERMAN, *Weak and strong uniform consistency of Kernel regression estimates*, Z. Wahrscheinlichkutstheorie verw Gebite, 61 (1982), pp. 405-415.

[16] I. MATYAS, *Random optimization*, Automat. Remote Control, 26 (1965), pp. 244-251.

[17] D. MICHIE AND R. ROSS, *Experiments with the adaptive graph traverser*, Machine Intelligence 5, B. Meltzer and D. Michie, eds., Edinburgh University Press, North America, 1970, pp. 301-318.

[18] N. J. NILSSON, *Principles of Artificial Intelligence*, Tioga, Palo Alto, CA, 1980.

[19] P. D. A. SCHOFIELD, *Complete solution of the "eight-puzzle,"* Machine Intelligence 1, N. L. Collins and D. Michie, eds., Oliver and Boyd, 1967, pp. 125-133.

[20] E. SCHUSTER AND S. YAKOWITZ, *Contributions to the theory of nonparametric regression, with application to system identification*, Ann. Statist., 7 (1979), pp. 139-149.

[21] S. J. YAKOWITZ AND L. FISHER, *On sequential search for the maximum of an unknown function*, J. Math. Anal. Appl., 41 (1973), pp. 234-259.

[22] S. J. YAKOWITZ, *A statistical foundation for machine learning with application to go-moku*, Comput. Math. Appl., 17 (1989), pp. 1095-1102.

[23] S. J. YAKOWITZ AND W. LOWE, *Nonparametric Bandits*, submitted for publication, 1989.

[24] S. J. YAKOWITZ, S. LI, AND T. JAYAWARDENA, *Machine learning from dependent samples, with application to queueing networks*, working paper, Systems and Industrial Engineering Department, University of Arizona, Tucson, AZ, 1989.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 4, pp. 713-731, July 1990

ON GENERALIZED CROSS VALIDATION FOR TENSOR SMOOTHING SPLINES*

LARRY L. SCHUMAKER† AND FLORENCIO I. UTRERAS‡

Abstract. The natural tensor-product smoothing spline is one of the methods of choice for fitting noisy data given on a grid. A generalized cross-validation procedure for automatic selection of the smoothing parameter in the method is introduced. It is shown that as in the well-known univariate and thin plate spline cases, the method selects the parameter in an asymptotically optimal way. Computational aspects of the method are also discussed, and a numerical example is presented. The method developed here can also be extended to complete and periodic tensor-product smoothing splines.

Key words. smoothing splines, tensor splines, cross validation, fitting noisy data

AMS(MOS) subject classification. 41A15

1. Introduction.

It is well known that the natural smoothing spline is an effective tool for fitting data in one variable, particularly if the method of generalized cross validation is used to select the smoothing parameter. Besides being easy to compute (cf. [3], [7], [11], [12], [16], [17], [18], [23], [27]), it has been shown that the method possesses a certain asymptotic optimality (cf. [3], [22]). Recently, we have developed a similar generalized cross-validation method for fitting univariate data using complete splines [14]. As in the natural case, we have shown that the method is asymptotically optimal, and that the optimal smoothing parameter can be computed efficiently.

In this paper we are interested in fitting surfaces to data given at points on a rectangular grid. While any of a variety of general fitting methods could be used, for example, thin plate smoothing splines [4]-[6], [16], [20]-[22], for the sake of efficiency it is certainly more reasonable to use tensor-product splines as discussed in [1].

The purpose of this paper is to provide a generalized cross-validation method for selecting the smoothing parameters when fitting gridded data using tensor-product splines. In particular, we shall treat both the computational and theoretical aspects of the method, including a proof of its asymptotic optimality.

The paper is organized as follows. In § 2 we give a general treatment of the method. Computational aspects are discussed in § 3. Sections 4 and 5 are devoted to an asymptotic analysis of the method, culminating in Theorem 5.6, which shows that the generalized cross-validation procedure introduced here produces optimal smoothing parameters (in a certain statistical sense). Section 6 contains a numerical example. Finally, § 7 is devoted to several remarks, including a discussion of how the results can be extended to complete and periodic tensor splines.

2. Tensor-product smoothing splines.

A general treatment of abstract tensor-product smoothing splines (in the bivariate case) has recently been given in [1]. For our purposes here, we need a somewhat different treatment of the subject. We carry out our discussion for the general case of d variables.

* Received by the editors October 31, 1988; accepted for publication (in revised form) May 9, 1989.

† Department of Mathematics, Vanderbilt University, Nashville, Tennessee 37235. This research was supported by the Deutsche Forschungsgemeinschaft and by National Science Foundation grant DMS-8602337.

‡ Department of Applied Mathematics, University of Chile, Casilla 5272 Correo 3, Santiago, Chile. This research was supported in part by Fondo Nacional de Ciencia Tecnología under grant 1296/86.

Let $\mathbf{n}=\{n_1, n_2, \cdots, n_d\}$ and $\mathbf{m}=\{m_1, m_2, \cdots, m_d\}$ be vectors of positive integers with $n_i \geqq 2m_i$ for $i=1,2,\cdots,d$. For each such i, let $[a_i, b_i] \subset \mathbb{R}$, and let

$$P_i=\{a_i=x_1^i<x_2^i<\cdots<x_{n_i}^i=b_i\} \tag{2.1}$$

be a partition of $[a_i, b_i]$. Let $\Omega \subset \mathbb{R}^d$ be the parallelopiped defined by the Cartesian product of the $[a_i, b_i]$; i.e.,

$$\Omega=\prod_{i=1}^d [a_i, b_i]. \tag{2.2}$$

Suppose that we are given measurements

$$z_{i_1,i_2,\cdots,i_d}=f(x_{i_1}^1, x_{i_2}^2, \cdots, x_{i_d}^d)+\varepsilon_{i_1,i_2,\cdots i_d}, \qquad 1 \leqq i_j \leqq n_j, \qquad j=1,\cdots,d, \tag{2.3}$$

where $f: \Omega \mapsto \mathbb{R}$ is the function we are trying to fit, and where the measurement errors $\varepsilon_{i_1,i_2,\cdots,i_d}$ are realizations of independent identically distributed random variables with zero mean and common variance σ^2. As already mentioned in the introduction, in this gridded data situation, one of the most efficient and effective ways of fitting the function f is to use tensor-product splines. To introduce them, we need some further notation.

For each $i=1,\cdots,d$, let

$$\mathscr{H}_i := H^{m_i}[a_i, b_i] := \{u: [a_i, b_i] \mapsto \mathbb{R}: u, u', \cdots, u^{(m_i)} \in L^2[a_i, b_i]\}. \tag{2.4}$$

These are Hilbert spaces, and taking their tensor product, we obtain a Hilbert space (cf. [13])

$$\mathscr{H}=\mathscr{H}_1 \otimes \mathscr{H}_2 \otimes \cdots \otimes \mathscr{H}_d, \tag{2.5}$$

which is the completion of the set

$$\mathscr{E}=\operatorname{span}\{u_1 \otimes u_2 \otimes \cdots \otimes u_d : u_i \in \mathscr{H}_i, \qquad i=1,\cdots,d\}, \tag{2.6}$$

with respect to the inner product defined on $\mathscr{E}$ by bilinear extension:

$$\langle u_1 \otimes u_2 \otimes \cdots \otimes u_d, v_1 \otimes v_2 \otimes \cdots \otimes v_d \rangle=\prod_{i=1}^d \langle u_i, v_i \rangle_{\mathscr{H}_i}, \tag{2.7}$$

where for each i, $\langle\, ,\rangle_{\mathscr{H}_i}$ is the inner product on $\mathscr{H}_i$.

Suppose $\lambda>0$ is prescribed. Fix $i=1,\cdots,d$. Then given $u, v \in \mathscr{H}_i$ and $w \in \mathbb{R}^{n_i}$, let

$$J_i(u) := Q_i(u,u)-2L_i(w,u), \tag{2.8}$$

where

$$Q_i(u,v)=\lambda \int_a^b u^{(m_i)}(t) v^{(m_i)}(t)\, dt+\frac{1}{n_i} \sum_{j=1}^{n_i} u(x_j^i) v(x_j^i) \tag{2.9}$$

and

$$L_i(w,u)=\frac{1}{n_i} \sum_{j=1}^{n_i} u(x_j^i) w_j. \tag{2.10}$$

It is well known (cf. [3], [18]) that there is a unique function $s_\lambda^{[i]} \in \mathscr{H}_i$ that minimizes J_i over $\mathscr{H}_i$. It is the univariate smoothing spline associated with the data w. Since $s_\lambda^{[i]}$ depends linearly on w, there exists a bounded linear operator $S_\lambda^{[i]}: \mathbb{R}^{n_i} \mapsto \mathscr{H}_i$ such that

$$S_\lambda^{[i]} w=s_\lambda^{[i]}. \tag{2.11}$$

Our aim now is to construct a certain quadratic form on $\mathcal{H}$ using the $J_1, \cdots, J_d$. First, let $Q = Q_1 \otimes Q_2 \otimes \cdots \otimes Q_d$ be the bilinear form

$$Q : \mathcal{H} \times \mathcal{H} \mapsto \mathbb{R}$$

defined by

$$Q(u_1 \otimes u_2 \otimes \cdots \otimes u_d, v_1 \otimes v_2 \otimes \cdots \otimes v_d) = \prod_{i=1}^{d} Q_i(u_i, v_i), \tag{2.12}$$

and extended by linearity to $\mathscr{E}$ and by density to $\mathcal{H}$. Similarly, we define $L : \bigotimes_{i=1}^{d} \mathbb{R}^{n_i} \times \mathcal{H} \mapsto \mathbb{R}$ by

$$L = \bigotimes_{i=1}^{d} L_i. \tag{2.13}$$

Now, for each $u \in \mathcal{H}$ and $z \in \bigotimes_{i=1}^{d} \mathbb{R}^{n_i}$, let

$$J(u) = Q(u, u) - 2L(z, u). \tag{2.14}$$

The quantity $J(u)$ equals, up to an additive constant, a combination of the residual sum of squares plus a penalty term depending on λ; for the bivariate case, see (2.35)-(2.36).

THEOREM 2.1. *For each $\lambda > 0$, there exists a unique function $s_\lambda \in \mathcal{H}$ that minimizes $J(u)$ over $\mathcal{H}$. This function is called the tensor-product smoothing spline.*

Proof. By the Lax-Milgram lemma, it suffices to prove that $L_z : u \mapsto L(z, u)$ is bounded, and that Q is coercive. The boundedness of L_z follows from the fact that each of the L_i is bounded as an operator from $\mathbb{R}^{n_i}$ to $\mathbb{R}$, while

$$|L(z, u)| \geqq \prod_{i=1}^{d} \|L_i\| \, \|z\| \, \|u\|, \tag{2.15}$$

(see [13]).

We turn now to the proof of the coerciveness of Q. First note that there exist $\alpha_1, \cdots, \alpha_d > 0$ such that

$$Q_i(u, u) \geqq \alpha_i \|u\|^2_{\mathcal{H}_i}. \tag{2.16}$$

Now if for each $i = 1, \cdots, d$, $\{\psi^i_j\}_{j=1}^{\infty}$ is an orthonormal set on $\mathcal{H}_i$, then $\{\bigotimes_{i=1}^{d} \psi^i_j\}_{j=1}^{\infty}$ is an orthonormal set in $\mathcal{H}$ (cf. [13]). But

$$\begin{aligned} &Q\left(\sum_{i_1 i_2, \cdots, i_d = 1}^{M} \beta_{i_1, i_2, \cdots, i_d} \psi^1_{i_1} \otimes \cdots \otimes \psi^d_{i_d}, \sum_{j_1, j_2, \cdots, j_d = 1}^{M} \gamma_{j_1 j_2, \cdots, j_d} \psi^1_{j_1} \otimes \cdots \otimes \psi^d_{j_d} \right) \\ &\qquad = \sum_{\substack{i_1, i_2, \cdots, i_d = 1 \\ j_1, j_2, \cdots, j_d = 1}}^{M} \beta_{i_1, i_2, \cdots, i_d} \gamma_{j_1, j_2, \cdots, j_d} \prod_{k=1}^{d} Q_k(\psi^k_{i_k}, \psi^k_{j_k}). \end{aligned} \tag{2.17}$$

Thus, the bilinear form (2.17) defined on $\bigotimes_{i=1}^{d} \mathbb{R}^M \times \bigotimes_{i=1}^{d} \mathbb{R}^M$ corresponds to the matrix

$$\mathcal{Q}_M = \bigotimes_{k=1}^{d} \mathcal{Q}_{k,M},$$

where

$$\mathcal{Q}_{k,M} = [Q_k(\psi^i_k, \psi^j_k)]_{i,j=1}^{M}. \tag{2.18}$$

Since the eigenvalues of a tensor product of matrices are the products of the eigenvalues of the individual matrices (cf. [13]), and since the eigenvalues of $\mathcal{Q}_{k,M}$ are bounded below by α_k, $k=1,\cdots,d$, we conclude that the eigenvalues of $\mathcal{Q}_M$ are bounded below by $\prod_{i=1}^d \alpha_i$. It follows that

$$\begin{aligned} Q\Big(\sum_{i_1,i_2,\cdots,i_d} \beta_{i_1,i_2,\cdots,i_d}\psi^1_{i_1}\otimes\cdots\otimes\psi^d_{i_d},\ \sum_{i_1,i_2,\cdots,i_d} \beta_{i_1,i_2,\cdots,i_d}\psi^1_{i_1}\otimes\cdots\otimes\psi^d_{i_d}\Big) \\ \geqq \prod_{i=1}^d \alpha_i \Big\| \sum_{i_1,i_2,\cdots,i_d} \beta_{i_1,i_2,\cdots,i_d}\psi^1_{i_1}\otimes\cdots\otimes\psi^d_{i_d}\Big\|^2. \end{aligned} \tag{2.19}$$

Now since span $\{\otimes_{i=1}^d \psi_j^i\}_{j=1}^\infty$ is dense in $\mathcal{H}$, we obtain the coercivity of Q:

$$Q(u,u)\geqq\Big(\prod_{i=1}^d \alpha_i\Big)\|u\|^2. \qquad \square$$

The following theorem gives a characterization of the tensor smoothing spline.

THEOREM 2.2. *The unique tensor smoothing spline* s_λ *of Theorem* 2.1 *can be written as*

$$s_\lambda = S_\lambda z, \tag{2.20}$$

where

$$S_\lambda = S_\lambda^{[1]}\otimes\cdots\otimes S_\lambda^{[d]}. \tag{2.21}$$

Proof. From the preceding theorem and the Lax-Milgram lemma, we conclude that the condition

$$Q(s_\lambda, v) = L(z,v), \qquad \forall v\in\mathcal{H}, \tag{2.22}$$

characterizes s_λ. Hence, there exists a linear bounded operator S_λ such that $s_\lambda = S_\lambda z$, and

$$\|S_\lambda\| \leqq \frac{\|L\|}{\prod_{i=1}^d \alpha_i}. \tag{2.23}$$

Now let $\{e_j^k\}_{j=1}^{n_k}$ be the canonical basis for $\mathbb{R}^{n_k}$. Then a basis for $\otimes_{k=1}^d \mathbb{R}^{n_k}$ is given by $\{\otimes_{k=1}^d e^k_{i_k}\}$, and z can be written as

$$z=\sum_{i_1=1}^{n_1}\sum_{i_2=1}^{n_2}\cdots\sum_{i_d=1}^{n_d} \hat{z}_{i_1,i_2,\cdots,i_d} e^1_{i_1}\otimes\cdots\otimes e^d_{i_d}.$$

Then

$$\bar{s}_\lambda = S_\lambda^{[1]}\otimes\cdots\otimes S_\lambda^{[d]}z=\sum_{i_1=1}^{n_1}\sum_{i_2=1}^{n_2}\cdots\sum_{i_d=1}^{n_d} \hat{z}_{i_1,i_2,\cdots,i_d} S_\lambda^{[1]}e^1_{i_1}\otimes\cdots\otimes S_\lambda^{[d]}e^d_{i_d}.$$

Now using the orthogonal basis defined earlier, we have

$$\begin{aligned} Q(\bar{s}_\lambda, \psi^1_{j_1}\otimes\cdots\otimes\psi^d_{j_d}) &= \sum_{i_1=1}^{n_1}\sum_{i_2=1}^{n_2}\cdots\sum_{i_d=1}^{n_d} \hat{z}_{i_1,i_2,\cdots,i_d}\prod_{k=1}^d Q_k(S_\lambda^{[k]}e^k_{i_k},\psi^k_{j_k}) \\ &= \sum_{i_1=1}^{n_1}\sum_{i_2=1}^{n_2}\cdots\sum_{i_d=1}^{n_d} \hat{z}_{i_1,i_2,\cdots,i_d}\prod_{k=1}^d L_k(e^k_{i_k},\psi^k_{j_k}) \\ &= L(z,\psi^1_{j_1}\otimes\cdots\otimes\psi^d_{j_d}). \end{aligned}$$

By the density of the orthonormal basis, we conclude that

$$Q(\bar{s}_\lambda, v) = L(z,v), \qquad \forall v\in\mathcal{H}.$$

Since the solution of (2.22) is unique, it follows that $\bar{s}_\lambda = s_\lambda$. $\square$

Theorem 2.2 asserts that the tensor smoothing spline can be calculated by computing the tensor product of the univariate smoothing operators. In the next section we will show how this can be accomplished in practice. In the remainder of this section we explore in more detail the connection with [9], [10]. For each $i=1,\cdots,d$, suppose we take

$$\langle u,v\rangle_{\mathcal{H}_i}=\int_{a_i}^{b_i}[u^{(m_i)}(t)v^{(m_i)}(t)+u(t)v(t)]\,dt \tag{2.24}$$

to be the inner product on $\mathcal{H}_i$. Now let

$$\mathcal{H}'=\{f\colon\Omega\mapsto\mathbb{R}\colon D^\alpha f\in L^2(\Omega),\quad \alpha_i\leqq m_i,\quad i=1,\cdots,d\}, \tag{2.25}$$

where D^α is the usual multi-index notation for partial derivates. We endow $\mathcal{H}'$ with the inner product

$$\langle f,g\rangle_{\mathcal{H}'}=\sum_{\alpha\in C_d}\int_\Omega D^\alpha f D^\alpha g, \tag{2.26}$$

where $C_d=\prod_{i=1}\{0,m_i\}$ represents the set of vertices of the "d-cube" in $\mathbb{R}^d$.

Now let $u_i, v_i\in\mathcal{H}_i$, $i=1,\cdots,d$. Then, clearly, if $f=\otimes_{i=1}^d u_i$ and $g=\otimes_{i=1}^d v_i$, the fact that $\alpha_i\leqq m_i$ and $u_i^{(\alpha_i)}\in L^2[a_i,b_i]$, $i=1,\cdots,d$, implies

$$\int_\Omega |D^\alpha f|^2=\prod_{i=1}^d\int_{a_i}^{b_i}[u_i^{(\alpha_i)}(t)]^2\,dt<\infty.$$

Moreover,

$$\begin{aligned}\langle f,g\rangle_{\mathcal{H}'}&=\sum_{\alpha\in C_d}\prod_{i=1}^d\int_{a_i}^{b_i}u_i^{(\alpha_i)}(t)v_i^{(\alpha_i)}(t)\,dt\\&=\prod_{i=1}^d\left[\int_{a_i}^{b_i}u_i^{(m_i)}(t)v_i^{(m_i)}(t)\,dt+\int_{a_i}^{b_i}u_i(t)v_i(t)\,dt\right]\\&=\prod_{i=1}^d\langle u_i,v_i\rangle.\end{aligned}$$

It follows that the inner product defined on the set $\mathcal{E}$ defined in (2.6) is the restriction of $\langle,\rangle_{\mathcal{H}'}$ to $\mathcal{E}$. Thus, $\mathcal{H}$ is a subset of $\mathcal{H}'$. We now prove that $\mathcal{H}=\mathcal{H}'$.

As seen in the proof of Theorem 2.1, $\{\psi_{i_1}^1\otimes\cdots\otimes\psi_{i_d}^d\}$ is an orthonormal basis for $\mathcal{H}$. Thus to show that $\mathcal{H}=\mathcal{H}'$, it suffices to show that

$$\langle f,\psi_{i_1}^1\otimes\cdots\otimes\psi_{i_d}^d\rangle_{\mathcal{H}'}=0$$

implies $f\equiv 0$, or equivalently, that $\{\psi_{i_1}^1\otimes\cdots\otimes\psi_{i_d}^d\}$ is also a basis for $\mathcal{H}'$. Now

$$\langle f,\psi_{i_1}^1\otimes\cdots\otimes\psi_{i_d}^d\rangle_{\mathcal{H}'}=\sum_{\alpha\in C_d}\int_\Omega D^\alpha f\prod_{k=1}^d D^{\alpha_k}\psi_k,$$

where for ease of notation, we now write ψ_k for $\psi_{i_k}^k$. Let

$$\begin{aligned}\omega^{(d-1)}(x_1,\cdots,x_{d-1})=\int_{a_d}^{b_d}\Bigg[&\frac{\partial^{m_d}f(x_1,\cdots,x_{d-1},t)}{\partial t^{m_d}}\psi_d^{(m_d)}(t)\\&+f(x_1,\cdots,x_{d-1},t)\psi_d(t)\Bigg]\,dt\end{aligned} \tag{2.27}$$

and

$$\omega^{(l)}(x_1,\cdots,x_l)=\int_{a_{l+1}}^{b_{l+1}}\left[\frac{\partial^{m_l}\omega^{(l+1)}(x_1,\cdots,x_l,t)}{\partial t^{m_l}}\psi_{l+1}^{(m_l)}(t)+\omega^{(l+1)}(x_1,\cdots,x_l,t)\psi_{l+1}(t)\right]dt \tag{2.28}$$

for $l=d-2,\cdots,1$. Then

$$\langle f,\psi_1\otimes\cdots\otimes\psi_d\rangle_{\mathcal{H}'}=\langle\omega^{(1)},\psi_1\rangle_{\mathcal{H}_1}=0. \tag{2.29}$$

This equality holds for any ψ_1 in the orthonormal basis $\{\psi_1^{i_1}\}_1^\infty$ of $\mathcal{H}_1$, and hence $\omega^{(1)}(x_1)=0$ for all x_1.

Using (2.28) for $l=1$, we get

$$0=\int_{a_2}^{b_2}\left[\frac{\partial^m\omega^{(2)}}{\partial t^m}(x_1,t)\psi^{(m)}(t)+\omega^{(2)}(x_1,t)\psi_2(t)\right]dt$$

for any $\psi_2\in\{\psi_2^{i_2}\}_1^\infty$, the orthonormal basis of $\mathcal{H}_2$. Hence, $\omega^{(2)}(x_1,x_2)=0$ for all x_1, x_2. By induction, it is easy to show that

$$f(x_1,\cdots,x_d)=0,\qquad \forall(x_1,\cdots,x_d)\in\Omega=\prod_{i=1}^d[a_i,b_i].$$

We have thus shown that $\mathcal{H}=\mathcal{H}'$. A similar proof leads to the following expression for Q:

$$Q(u,u)=\sum_{\alpha\in C_d}\lambda^{p(\alpha)}\int_\Omega|D^\alpha u|^2\,d\mu_\alpha, \tag{2.30}$$

where

$$p(\alpha)=\sum_{i=1}^d\frac{\alpha_i}{m_i},\qquad d\mu_\alpha=d\mu_{\alpha_1}\otimes d\mu_{\alpha_2}\otimes\cdots\otimes d\mu_{\alpha_d} \tag{2.31}$$

and

$$\int_{a_i}^{b_i}\phi(x)\,d\mu_{\alpha_i}(x)=\begin{cases}\int_{a_i}^{b_1}\phi(x)\,dx, & \alpha_i=m_i\\ \frac{1}{n_i}\sum_{j=1}^{n_i}\phi(x_j^i), & \alpha_i=0.\end{cases} \tag{2.32}$$

Similarly, we have

$$L(z,u)=\sum_{i_1,i_2,\cdots,i_d}z_{i_1,i_2,\cdots,i_d}u(x_{i_1}^1,x_{i_2}^2,\cdots,x_{i_d}^d). \tag{2.33}$$

Now let $d=2$. In this case, $C_d=\{(m_1,m_2),(m_1,0),(0,m_2),(0,0)\}$ and we get

$$\begin{aligned}Q(u,u)&=\lambda^2\int_{a_1}^{b_1}\int_{a_2}^{b_2}\left(\frac{\partial^4u(\alpha,\beta)}{\partial\alpha^2\partial\beta^2}\right)^2d\alpha\,d\beta+\lambda\frac{1}{n_2}\sum_{j=1}^{n_2}\int_{a_1}^{b_1}\left[\frac{\partial^2u(\alpha,x_j^2)}{\partial\alpha^2}\right]^2d\alpha\\&\quad+\lambda\frac{1}{n_1}\sum_{i=1}^{n_1}\int_{a_2}^{b_2}\left[\frac{\partial^2u(x_i^1,\beta)}{\partial\beta^2}\right]^2d\beta+\frac{1}{n_1n_2}\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}[u(x_i^1,x_j^2)]^2,\end{aligned} \tag{2.34}$$

which corresponds to the expression given in [9]. Thus in this case the smoothing spline is the minimizer of the quantity

$$J(u)=RSS(u)+PEN(u)-\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}z_{ij}^2, \tag{2.35}$$

where

$$RSS(u) = \frac{1}{n_1 n_2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} (u(x_i^1, x_j^2) - z_{ij})^2 \tag{2.36}$$

is the residual sum of squares and $PEN(u)$ is the penalty given by the first three terms of (2.34) and $\sum\sum z_{ij}^2/n_1n_2$ is a constant. Gu et al. [8] have studied a tensor smoothing spline in $\mathcal{H}$ with a different penalty and which allows scattered data, but of course it lacks the tensor structure of the smoothing spline presented here.

3. Computational aspects and generalized cross validation. We begin this section by showing that the tensor smoothing spline can be computed by solving certain linear systems of equations, and that this computation can be organized in a highly efficient manner. First, we recall that for each $1 \leqq i \leqq d$, the univariate smoothing spline corresponding to data $y \in \mathbb{R}^{n_i}$ measured at points $a_i = x_1^i < x_2^i < \cdots < x_{n_i}^i = b_i$ can be written as

$$s_\lambda^{[i]} = S_\lambda^{[i]} y = W_i A_\lambda^{[i]} y, \tag{3.1}$$

where $A_\lambda^{[i]}$ is the so-called influence matrix for univariate spline smoothing (cf. [3], [17]), $W_i = (\theta_1^{[i]}, \cdots, \theta_{n_i}^{[i]})^T$, and $\{\theta_j^{[i]}\}_{j=1}^{n_i}$ are the cardinal natural splines of degree $2m_i - 1$ satisfying

$$\theta_j^{[i]}(x_k^i) = \begin{cases} 1, & j = k \\ 0, & j \neq k. \end{cases} \tag{3.2}$$

Using well-known properties of tensor products (cf. [2]), for $z \in \bigotimes_{i=1}^d \mathbb{R}^{n_i}$ we have

$$S_\lambda z = \bigotimes_{i=1}^d S_\lambda^{[i]} z = \bigotimes_{i=1}^d W_i A_\lambda^{[i]} z = \left(\bigotimes_{i=1}^d W_i \right) \left(\bigotimes_{i=1}^d A_\lambda^{[i]} \right) z. \tag{3.3.}$$

The first tensor product in the last expression in (3.3) gives a mapping from $\bigotimes_{i=1}^d \mathbb{R}^{n_i}$ into $\bigotimes_{i=1}^d \mathcal{H}_i$ defined by

$$\left(\bigotimes_{i=1}^d W_i \right) r = \sum_{i_1=1}^{n_1} \sum_{i_2=1}^{n_2} \cdots \sum_{i_d=1}^{n_d} r_{i_1, i_2, \cdots, i_d} \theta_{i_1}^{[1]} \otimes \cdots \otimes \theta_{i_d}^{[d]}. \tag{3.4}$$

We now discuss the second tensor product in the last expression in (3.3). Let I_{n_i} denote the identity matrix in $\mathbb{R}^{n_i}$. By elementary properties of tensor products, we have

$$\begin{aligned} \bigotimes_{i=1}^d A_\lambda^{[i]} = {} & (I_{n_1} \otimes I_{n_2} \otimes \cdots \otimes I_{n_{d-1}} \otimes A_\lambda^{[d]}) \circ (I_{n_1} \otimes \cdots \otimes I_{n_{d-2}} \otimes A_\lambda^{[d-1]} \otimes I_{n_d}) \\ & \circ \cdots \circ (A_\lambda^{[1]} \otimes I_{n_2} \otimes \cdots \otimes I_{n_d}). \end{aligned} \tag{3.5}$$

Using de Boor's cyclic-shift factorization technique [2], we can rewrite this as

$$\begin{aligned} \bigotimes_{i=1}^d A_\lambda^{[i]} = {} & P_d(A_\lambda^{[d]} \otimes I_{n_1} \otimes \cdots \otimes I_{n_{d-1}}) \circ P_{d-1}(A_\lambda^{[d-1]} \otimes I_{n_d} \otimes I_{n_1} \otimes \cdots \otimes I_{n_{d-2}}) \\ & \circ \cdots \circ P_1(A_\lambda^{[1]} \otimes I_{n_2} \otimes \cdots \otimes I_{n_d}), \end{aligned} \tag{3.6}$$

where P_l denotes the permutation that shifts indices circularly to the left; i.e.,

$$P_l(B_1 \otimes \cdots \otimes B_d) = B_{d-l} \otimes B_{d-l+1} \otimes \cdots \otimes B_d \otimes B_1 \otimes \cdots \otimes B_{d-l-1}.$$

From (3.6), we see that the effect of applying $\bigotimes_{i=1}^d A_\lambda^{[i]}$ to z amounts to successively applying a transformation of the form

$$B_1 \otimes I_{m_2} \otimes \cdots \otimes I_{m_d}, \tag{3.7}$$

and then permuting the indices. But the application of (3.7) to z is the same as applying B_1 to the first index of $z_{i_1,i_2,\cdots,i_d} \in \mathbb{R}^{m_1}$ while holding all other indices constant. This corresponds to smoothing the data in the i_1 direction, keeping all other indices fixed. Thus we have shown that computing the tensor smoothing spline simply amounts to successively solving univariate smoothing problems, and that the available univariate software (cf. [11], [12]) for solving them can be immediately applied.

In the remainder of this section we introduce and discuss the computation of a suitable cross-validation function that can be used to automatically select an optimal value for the smoothing parameter λ for tensor smoothing. For each $\lambda > 0$, let

(3.8) $$V(\lambda) = \frac{(1/N)\sum_{i_1,i_2,\cdots,i_d}[z_{i_1,i_2,\cdots,i_d} - s_\lambda(x^1_{i_1}, \cdots, x^d_{i_d})]}{[1-(1/N)\operatorname{Tr}(\otimes_{i=1}^d A_\lambda^{[i]})]^2},$$

where $N = \prod_{i=1}^d n_i$. In analogy with the univariate case (cf. [3], [18]) we call $V(\lambda)$ the *generalized cross-validation* (GCV) function. We denote the first point on $(0, \infty)$ where V assumes a minimum by λ^V. In § 4 we show that this method of choosing the smoothing parameter is asymptotically optimal.

We turn now to the problem of computing λ^V. As in the univariate case [3], [16]-[18], this has to be done by evaluating $V(\lambda)$ at several selected points λ_s and then selecting the best value by a search procedure (such as the "golden search" method, cf. [16], [18]). For the evaluation of $V(\lambda)$, we note that the eigenvalues of $\otimes_{i=1}^d A_\lambda^{[i]}$ are given by the product of the eigenvalues of the matrices $A_\lambda^{[i]}$, and thus

(3.9) $$\operatorname{Tr}\left(\bigotimes_{i=1}^d A_\lambda^{[i]}\right) = \prod_{i=1}^d \operatorname{Tr}(A_\lambda^{[i]}).$$

Then writing

(3.10) $$\|u\|^2 = \sum_{i_1,i_2,\cdots,i_d} u^2_{i_1,i_2,\cdots,i_d},$$

we get

(3.11) $$V(\lambda) = \frac{(1/N)\|z - \otimes_{i=1}^d A_\lambda^{[i]} z\|^2}{[1-(1/N)\prod_{i=1}^d \operatorname{Tr}(A_\lambda^{[i]})]^2}.$$

The numerator in (3.11) is simply the sum of squares of the differences between the given data values $z_{i_1,i_2,\cdots,i_d}$ and the values of the smoothing spline at the points $x_{i_1,i_2,\cdots,i_d}$ as $x_{i_1,i_2,\cdots,i_d}$ runs over the grid. Efficient methods for computing the traces appearing in the denominator of (3.11) have been discussed in several papers [11], [17], [18]. For the case of equally spaced data, approximate values for the eigenvalues (cf. [17]) can be used, while in the unequally spaced case, the algorithm given in [11] and generalized in [14] can be applied.

4. Properties of the true error and the validation function. In this section we collect several results that will be needed in the following section to show that the generalized cross-validation method is asymptotically optimal.

We begin with some notation. Throughout this section we suppose that $f \in \mathcal{H}$ is a fixed unknown function that we are trying to fit. Let $\mathbf{n} = (n_1, \cdots, n_d)$ and $\mathbf{m} = (m_1, \cdots, m_d)$ be vectors of positive integers with $n_i \geqq 2m_i$ for $i = 1, \cdots, d$. Let

(4.1) $$P = P_1 \otimes \cdots \otimes P_d$$

be a partition of the parallelopiped Ω defined in (2.2), where the P_i are as in (2.1). Suppose that $\mathbf{z} \in \otimes_{i=1}^d \mathbb{R}^{n_i}$ is the vector whose components are the set of measurement values in (2.3), arranged in their nature lexicographical order. Given a smoothing parameter $\lambda > 0$, let $s_\lambda = s_{\mathbf{m},\mathbf{n},P,\mathbf{z},\lambda}$ denote the corresponding tensor natural smoothing spline of Theorem 2.1.

We are interested in how well s_λ fits f. To measure this, we introduce the *true mean squared error*

$$T(\lambda) = \frac{1}{n_1 n_2 \cdots n_d} \sum_{i_1,i_2,\cdots,i_d} [f(x_{i_1}^1, \cdots, x_{i_d}^d) - s_\lambda(x_{i_1}^1, \cdots, x_{i_d}^d)]^2. \tag{4.2}$$

This expression can be rewritten as

$$T(\lambda) = \|A_\lambda \mathbf{z} - \mathbf{f}\|^2, \tag{4.3}$$

where $\mathbf{f}$ is the vector whose components are the values $f(x_{i_1}^1, \cdots, x_{i_d}^d)$ arranged in lexicographical order, and A_λ is the matrix

$$A_\lambda = \bigotimes_{i=1}^{d} A_\lambda^{[i]}. \tag{4.4}$$

Note that $\mathbf{z} = \mathbf{f} + \mathbf{e}$, where $\mathbf{e}$ is the vector whose components are the values $e_{i_1,i_2,\cdots,i_d}$ arranged in lexicographical order.

As in § 1, we now assume that the $\varepsilon_{i_1,i_2,\cdots,i_d}$ are realizations of independent identically distributed random variables with zero means and common variance σ^2. Thus, $T(\lambda)$ is also a random variable, and we may take its expected value. As in [3], it is easy to see that the result is

$$E(T(\lambda)) = \sigma^2 \mu_2(\lambda) + b^2(\lambda), \tag{4.5}$$

where

$$\mu_2(\lambda) = \frac{1}{n_1 n_2 \cdots n_d} \operatorname{Tr}[A_\lambda^2] = \prod_{i=1}^{d} \left[\frac{1}{n_i} \operatorname{Tr}(A_\lambda^{[i]})^2\right], \tag{4.6}$$

and

$$b^2(\lambda) = \frac{1}{n_1 n_2 \cdots n_d} \|\mathbf{f} - A_\lambda \mathbf{f}\|^2. \tag{4.7}$$

We shall also be interested in the expected value of the cross-validation function $V(\lambda)$ defined in (3.8). Again as in [3], it is easy to see that

$$E(V(\lambda)) = \frac{b^2(\lambda) + \sigma^2(1 - 2\mu_1(\lambda) + \mu_2(\lambda))}{(1 - \mu_1(\lambda))^2}, \tag{4.8}$$

where

$$\mu_1(\lambda) = \prod_{i=1}^{d} \left[\frac{1}{n_i} \operatorname{Tr}(A_\lambda^{[i]})\right]. \tag{4.9}$$

The following lemma follows as in [3].

LEMMA 4.1. *For all* $\lambda > 0$,

$$\frac{|E(T(\lambda)) - (E(V(\lambda)) - \sigma^2)|}{E(T(\lambda))} \leqq \Delta(\lambda), \tag{4.10}$$

where

$$\Delta(\lambda) = \left[\mu_1^2(\lambda) - 2\mu_1(\lambda) + \frac{\mu_1^2(\lambda)}{\mu_2(\lambda)}\right] \frac{1}{(1 - \mu_1^2(\lambda))^2}. \tag{4.11}$$

Lemma 4.1 asserts that, as in the univariate and thin plate spline cases [21], the behavior of $E(V(\lambda))$ and $E(T(\lambda))$ is controlled by the behavior of $b^2(\lambda)$, $\mu_1(\lambda)$, and $\mu_2(\lambda)$. We turn to a study of these quantities.

LEMMA 4.2. *Given a partition P, let*

$$B_{\mathbf{n},P} = \max_{1\leqq i\leqq d} \frac{\max_{1\leqq k\leqq n_i-1} |x^i_{k+1}-x^i_k|}{\min_{1\leqq k\leqq n_i-1} |x^i_{k+1}-x^i_k|}. \tag{4.12}$$

Then there exist constants L_1, U_1 and L_2, U_2 depending only on $\mathbf{m}$ *and $B_{\mathbf{n},P}$ such that for all $\lambda>0$ with $\lambda\leqq 1$ and $n_i\lambda^{1/2m_i}\geqq 2(m_i-1)$, $i=1,\cdots,d$,*

$$\frac{L_1}{\prod_{i=1}^d (n_i\lambda^{1/2m_i})} \leqq \mu_1(\lambda) \leqq \frac{U_1}{\prod_{i=1}^d (n_i\lambda^{1/2m_i})} \tag{4.13}$$

$$\frac{L_2}{\prod_{i=1}^d (n_i\lambda^{1/2m_i})} \leqq \mu_2(\lambda) \leqq \frac{U_2}{\prod_{i=1}^d (n_i\lambda^{1/2m_i})}. \tag{4.14}$$

Proof. It was shown in [21] that for each $i=1,\cdots,d$, there exist constants L^i_1, U^i_1 and L^i_2, U^i_2 such that for all λ satisfying the stated conditions,

$$\frac{L^i_1}{n_i\lambda^{1/2m_i}} \leqq \frac{\operatorname{Tr}(A^{[i]}_\lambda)}{n_i} \leqq \frac{U^i_1}{n_i\lambda^{1/2m_i}} \tag{4.15}$$

$$\frac{L^i_2}{n_i\lambda^{1/2m_i}} \leqq \frac{\operatorname{Tr}(A^{[i]}_\lambda)^2}{n_i} \leqq \frac{U^i_2}{n_i\lambda^{1/2m_i}}. \tag{4.16}$$

Multiplying together the inequalities in (4.15) for $i=1,\cdots,d$ yields (4.13). The proof of (4.14) follows from (4.16) in the same way. □

Our next lemma gives some information on the size of $b^2(\lambda)$.

LEMMA 4.3. *There exists a constant K depending only on* $\mathbf{m}$ *and the constant $B_{\mathbf{n},P}$ in* (4.12) *such that*

$$b^2(\lambda)\leqq K\lambda\,\|f\|^2_{\mathscr{H}}. \tag{4.17}$$

Proof. Let I_i be the $n_i\times n_i$ identity matrix, $i=1,\cdots,d$. Then

$$b^2(\lambda)=\frac{1}{\prod_{i=1}^d n_i}\,\|(I_1\otimes\cdots\otimes I_d - A^{[1]}_\lambda\otimes\cdots\otimes A^{[d]}_\lambda)\mathbf{f}\|^2. \tag{4.18}$$

But

$$\begin{aligned} I_1\otimes\cdots\otimes I_d - A^{[1]}_\lambda\otimes\cdots\otimes A^{[d]}_\lambda &= e_1(\lambda)\otimes I_2\otimes\cdots\otimes I_d \\ &\quad + A^{[1]}_\lambda\otimes e_2(\lambda)\otimes I_3\otimes\cdots\otimes I_d \\ &\quad +\cdots + A^{[1]}_\lambda\otimes\cdots\otimes A^{[d-1]}_\lambda\otimes e_d(\lambda), \end{aligned} \tag{4.19}$$

where

$$e_i(\lambda)=I_i-A^{[i]}_\lambda, \qquad i=1,\cdots,d. \tag{4.20}$$

From the univariate theory [3], we know that for all $g\in\mathscr{H}_i$,

$$\frac{1}{n_i}\|e_i(\lambda)\mathbf{g}\|^2\leqq\lambda\int_{a_i}^{b_i}[g^{(m_i)}(t)]^2\,dt\leqq\lambda\,\|g\|^2_{\mathscr{H}_i} \tag{4.21}$$

and

$$\frac{1}{n_i}\|A_\lambda^{[i]}\mathbf{g}\|^2 \leqq \frac{1}{n_i}\sum_{j=1}^{n_i}[g(x_j^i)]^2.$$

Moreover, from [3] we also have

$$\frac{1}{n_i}\|A_\lambda^{[i]}\mathbf{g}\|^2 \leqq C_m\|g\|_{\mathscr{H}_i}^2 \tag{4.22}$$

for all $i=1,\cdots,d$, where C_m depends only on the constant $B_{n,P}$ in (4.12) and the integer $m=\max\{m_1,\cdots,m_d\}$.

Now combining the above facts and using properties of tensor products, we get

$$\frac{b^2(\lambda)}{d} \leqq (\lambda+\lambda C_m+\cdots\lambda^{d-1}C_m^{d-1})\|f\|_{\mathscr{H}}^2 \leq d\lambda\frac{(1-C_m^d)}{(1-C_m)}\|f\|_{\mathscr{H}}^2, \tag{4.23}$$

which is (4.17) with $K=d(1-C_m^d)/(1-C_m)$. □

Lemma 4.3 asserts that $b^2(\lambda)$ goes to zero as $\lambda\to 0$. A kind of converse is given in Lemma 5.2 below. In preparation for the proof of that lemma, we devote the rest of this section to a discussion of the *Tikhonov regularizer* of a function f. For each $i=1,\cdots,d$, let $\tau_i(\delta)$ be the mapping from $\mathscr{H}_i$ to $\mathscr{H}_i$ such that $\tau_i(\delta)u$ is the unique solution of the problem of minimizing

$$\delta\int_{a_i}^{b_i}[v^{(m_i)}(t)\ ^2+v^2(t)-2u(t)v(t)]\,dt \tag{4.24}$$

over all $v\in\mathscr{H}_i$. Then the *tensor Tikhonov regularizer* of a function $f\in\mathscr{H}$ is given by

$$f_\delta=\tau_1(\delta)\otimes\cdots\otimes\tau_d(\delta)f. \tag{4.25}$$

The proof of the following result is identical to that of Theorem 2.2.

LEMMA 4.4. *For any $\delta>0$, f_δ is the unique minimizer of*

$$\left[\bigotimes_{i=1}^{d}\mathscr{H}_i\right](g,g)-2\left[\bigotimes_{i=1}^{d}\mathscr{L}_i\right](g,f), \tag{4.26}$$

where

$$\mathscr{H}_i(u,v)=\delta\int_a^b[u^{(m_i)}(t)v^{(m_i)}(t)+u(t)v(t)]\,dt$$

and

$$\mathscr{L}_i(u,v)=\int_{a_i}^{b_i}u(t)v(t)\,dt.$$

Moreover, f_δ is characterized by the equation

$$\mathscr{H}(f_\delta,u)=\mathscr{L}(f,u),\quad \text{for all } u\in\mathscr{H},$$

where

$$\mathscr{L}(u,v)=\int_\Omega uv$$

and

$$\mathscr{H}(u,v)=\prod_{i=1}^{d}\mathscr{H}_i(u,v)=\sum_{\alpha\in C_d}\delta^{p(\alpha)}\int_\Omega D^\alpha u D^\alpha v,$$

where $p(\alpha)$ is defined in (2.31).

5. Asymptotic optimality. In this section we are interested in how our smoothing method behaves as we take a sequence of finer and finer partitions of Ω. In particular, we suppose that $\mathbf{n}_p$, P_p describes a sequence of partitions of Ω for $p=1,2,\cdots$, and that $\mathbf{z}_p$ is the corresponding sequence of data vectors. Throughout this section we assume that the sequence of partitions is *quasi-uniform* in the sense that there is a fixed constant B such that for all $p>0$,

$$B_{n_p,P_p} \leqq B < \infty. \tag{5.1}$$

In addition, we suppose that

$$\min_{1\leqq i\leqq d} (\mathbf{n}_p)_i \to \infty \quad \text{as} \quad p\to\infty. \tag{5.2}$$

Note that we are holding fixed the function f and the vector $\mathbf{m}$ which determines the degrees of the spline in each of its variables. For ease of notation, we now make the following identification:

$$s_{p,\lambda} = s_{\mathbf{m},\mathbf{n}_p,\, P_{p,z_p},\, \lambda}. \tag{5.3}$$

For each p, let $T_p(\lambda)$ be the true mean-squared error defined as in (4.2), and let $V_p(\lambda)$ be the generalized cross-validation function defined as in (3.8). Let λ_p^V and λ_p^T be defined by

$$E[V_p(\lambda_p^V)] = \min_{\lambda>0} E[V_p(\lambda)] \tag{5.4}$$

and

$$E[T_p(\lambda_p^T)] = \min_{\lambda>0} E[T_p(\lambda)], \tag{5.5}$$

where, as in § 4, E denotes expected value. From the definitions, it is clear that

$$E[T_p(\lambda_p^T)] \leqq E[T_p(\lambda_p^V)].$$

In Theorem 5.6 below we show that the ratio of these two quantities approaches 1 as $p\to\infty$ so that for large values of p, the value of the smoothing parameter λ_p^V obtained from cross validation can be used as a good substitute for the optimal smoothing parameter λ_p^T. To prove this result, we need several preliminary results.

To get started, we first prove that for given $\delta>0$, the smoothing splines associated with exact measurements on a function f converge to the Tikhonov regularizer f_δ defined in (4.25).

LEMMA 5.1. *Suppose P_p is a sequence of quasi-uniform partitions satisfying* (5.2). *Let*

$$f_{p,\delta} = s_{\mathbf{m},\mathbf{n}_p,P_p,\mathbf{f}_p,\delta}.$$

Here $\mathbf{f}_p$ is the vector of true measurements on the function f. Then

$$\lim_{p\to\infty} f_{p,\delta} = f_\delta,$$

where f_δ is the Tikhonov regularizer defined in (4.25).

Proof. Let $\mathcal{I}_i : \mathcal{H}_i \mapsto \mathbb{R}^{n_i}$ be the evaluation operator defined by

$$\mathcal{I}_i v = (v(x_1^i),\cdots, v(x_{n_i}^i))^T.$$

Then

$$f_{p,\delta} = \bigotimes_{i=1}^{d} (S_\delta^{[i]}\mathcal{I}_i) f$$

and

$$\begin{aligned}f_\delta - f_{p,\delta} &= (\tau_\delta^{[1]}\otimes\cdots\otimes\tau_\delta^{[d]} - S_\delta^{[1]}\mathcal{I}_1\otimes\cdots\otimes S_\delta^{[d]}\mathcal{I}_d)f\\ &= [(\tau_\delta^{[1]} - S_\delta^{[1]}\mathcal{I}_1)\otimes\tau_\delta^{[2]}\cdots\otimes\tau_\delta^{[d]} + S_\delta^{[1]}\mathcal{I}_1\otimes(\tau_\delta^{[2]} - S_\delta^{[2]}\mathcal{I}_2)\otimes\tau_\delta^{[3]}\cdots\otimes\tau_\delta^{[d]}\\ &\quad+\cdots+S_\delta^{[1]}\mathcal{I}_1\otimes\cdots\otimes S_\delta^{[d-1]}\mathcal{I}_{d-1}\otimes(\tau_\delta^{[d]} - S_\delta^{[d]}\mathcal{I}_d)]f.\end{aligned}$$

Now from the univariate theory [3], [21],

$$\|\tau_\delta^{[i]} - S_\delta^{[i]}\mathcal{I}_i\| \leqq \mathcal{O}\left(\frac{1}{n_i}\right)$$

$$\|\tau_\delta^{[i]}\| \leqq (1+\delta)\sqrt{2}$$

and

$$\|S_\delta^{[i]}\mathcal{I}_i\| \leqq (1+C_{m_i}).$$

Combining these results leads to the desired assertion. □

We are now ready to prove a kind of converse to Lemma 4.3.

LEMMA 5.2. *Suppose P_p is a sequence of partitions as in Lemma* 5.1, *and that λ_p is a sequence of positive smoothing parameters. Suppose that f is not in the set $\mathcal{P}_{m_1-1}\otimes\cdots\otimes\mathcal{P}_{m_d-1}$, where, in general, $\mathcal{P}_r$ denotes the space of univariate polynomials of degree at most r. Then a necessary condition for the quantity $b_p^2(\lambda_p)$, defined as in* (4.7), *to go to zero is that $\lambda_p\to 0$.*

Proof. With $N = n_1\cdots n_d$, we have

$$\begin{aligned}b_p^2(\lambda_p) &= \frac{1}{N}\|\mathbf{f} - A_{\lambda_p}^{[1]}\otimes\cdots\otimes A_{\lambda_p}^{[d]}\mathbf{f}\|^2\\ &= \frac{1}{N}\sum_{i_1,i_2,\cdots,i_d}\left[1 - \frac{1}{(1+\lambda_p\lambda_{i_1}^1)\cdots(1+\lambda_p\lambda_{i_d}^d)}\right]\hat{f}^2_{i_1,i_2,\cdots,i_d},\end{aligned}$$

where $\hat{f}_{i_1,i_2,\cdots,i_d}$ are the Fourier coefficients of $\mathbf{f}$ in the basis of eigenvalues of $A_\lambda^{[1]}\otimes\cdots\otimes A_\lambda^{[d]}$, and $1/(1+\lambda_p\lambda_{i_k}^k)$ is the i_kth eigenvalue of $A_\lambda^{[k]}$. Thus

$$b^2(\lambda_p) = \frac{1}{N}\sum_{i_1,i_2,\cdots,i_d}\left[\frac{(1+\lambda_p\lambda_{i_1}^1)\cdots(1+\lambda_p\lambda_{i_d}^d)-1}{(1+\lambda_p\lambda_{i_1}^1)\cdots(1+\lambda_p\lambda_{i_d}^d)}\right]^2\hat{f}^2_{i_1,i_2,\cdots,i_d}$$

is a decreasing function of λ_p as λ_p decreases. It follows that if the sequence λ_p is bounded below by some constant $c>0$, then $b_p^2(\lambda_p)\geqq b_p^2(\delta)$.

But as $p\to\infty$, we have

$$\lim_{p\to\infty}\frac{1}{N}\|\mathbf{f} - \mathcal{I}_1\otimes\cdots\otimes\mathcal{I}_d f_{p,\delta}\|^2 = \int_\Omega (f-f_\delta)^2.$$

Thus

$$\lim_{p\to\infty} b_p^2(\lambda_p) = 0 \quad\text{implies}\quad \int_\Omega (f-f_\delta)^2 = 0,$$

and thus that $f_\delta \equiv f$ for all $\delta > 0$. This means that

$$(\tau_1(\delta)\otimes\cdots\otimes\tau_d(\delta))f = f$$

and so

$$\mathcal{Q}(f,f) = \mathcal{Q}(f_\delta, f) = \mathcal{L}(f,f).$$

We conclude that

$$\sum_{\alpha \in C_d \setminus \{0\}} \delta^{p(\alpha)} \int_\Omega |D^\alpha f|^2 = 0;$$

that is, $f \in \mathcal{P}_{m_1-1} \otimes \cdots \otimes \mathcal{P}_{m_d-1}$. This contradiction establishes the lemma. □

The following lemma follows directly from (4.5) coupled with Lemmas 4.2, 4.3, and 5.1 (cf. [21]).

LEMMA 5.3. *Suppose λ_p^T is the optimal smoothing parameter defined in* (5.5) *corresponding to a sequence of partitions P_p as in Lemma* 5.1. *Then*

$$\lim_{p \to \infty} \lambda_p^T = 0 \tag{5.6}$$

$$\lim_{p \to \infty} \prod_{i=1}^{d} [n_i (\lambda_p^T)^{1/2m_i}] = \infty \tag{5.7}$$

$$\lim_{p \to \infty} E[T_p(\lambda_p^T)] = 0. \tag{5.8}$$

The following lemma follows from (4.11), (4.13), and (4.14) as in [21].

LEMMA 5.4. *Given a sequence P_p of quasi-uniform partitions as in Lemma* 5.1, *let $\Delta_p(\lambda)$ be as defined in* (4.11). *Let λ_p be any sequence with*

$$\lim_{p \to \infty} \lambda_p = 0 \quad \textit{and} \quad \lim_{p \to \infty} \left[\prod_{i=1}^{d} [n_i (\lambda_p)^{1/2m_i}\right] = 0.$$

Then

$$\lim_{p \to \infty} \Delta_p(\lambda_p) = 0. \tag{5.9}$$

We now show that the sequence λ_p^V satisfies the same properties as those enunciated in Lemma 5.3 for λ_p^T.

LEMMA 5.5. *Suppose λ_p^V is the smoothing parameter defined in* (5.4) *corresponding to a sequence of partitions that are quasi-uniform and satisfy* (5.2). *Then*

$$\lim_{p \to \infty} \lambda_p^V = 0 \tag{5.10}$$

$$\lim_{p \to \infty} \prod_{i=1}^{d} [n_i (\lambda_p^V)^{1/2m_i}] = \infty \tag{5.11}$$

$$\lim_{p \to \infty} E[T_p(\lambda_p^V)] = 0. \tag{5.12}$$

Proof. Following the proof of Lemma 3.4 in [21], from (4.10) we can deduce that

$$E[V_p(\lambda_p^V)] - \sigma^2 \leqq E[V_p(\lambda_p^T)] - \sigma^2 \leqq E[T_p(\lambda_p^T)][1 + \Delta_p(\lambda_p^T)].$$

It follows that

$$\lim_{p \to \infty} E[V_p(\lambda_p^V) - \sigma^2] = 0.$$

But

$$E[V_p(\lambda_p^V)] - \sigma^2 = \frac{b_p^2(\lambda_p^V) + \sigma^2(\mu_{2,p}(\lambda_p^V) - \mu_{1,p}^2(\lambda_p^V))}{(1 - \mu_{1,p}(\lambda_p^V))^2},$$

and thus, since $\mu_{1,p}^2(\lambda_p^V) \leqq \mu_{2,p}(\lambda_p^V)$, $\lim_{p\to\infty} b_p^2(\lambda_p^V) = 0$. Now applying Lemma 5.2, we conclude that $\lim_{p\to\infty} \lambda_p^V = 0$. Moreover, we get

$$\lim_{p\to\infty} \frac{\mu_{2,p}(\lambda_p^V) - \mu_{1,p}^2(\lambda_p^V)}{(1-\mu_{1,p}^2(\lambda_p^V))^2} = 0. \tag{5.13}$$

This implies that

$$\frac{1}{N} \sum_{i_1,i_2,\cdots,i_d} [\beta_{i_1,i_2,\cdots,i_d}(\lambda_p^V) - \mu_{1,p}(\lambda_p^V)]^2 = \mu_{2,p}(\lambda_p^V) - \mu_{1,p}^2(\lambda_p^V) \to 0$$

as $p \to \infty$, where $\beta_{i_1,i_2,\cdots,i_d}(\lambda_p^V)$ are the eigenvalues of the matrix $A(\lambda_p^V)$. These eigenvalues are the products of the eigenvalues of the matrices $A_i(\lambda_p^V)$, $i = 1, \cdots, d$. Now following the proof in [21], we deduce that

$$\lim_{p\to\infty} \prod_{i=1}^{d} n_i(\lambda_p^V)^{1/2m_i} = \infty.$$

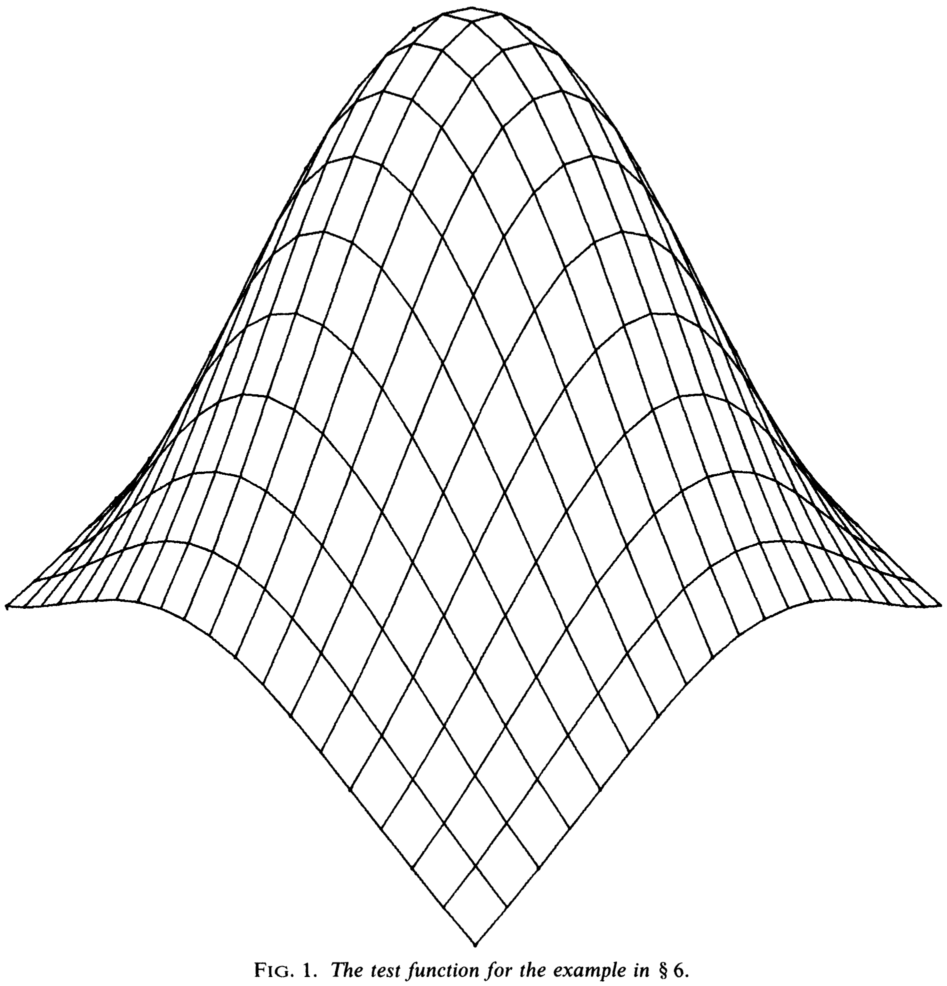


FIG. 1. *The test function for the example in* § 6.

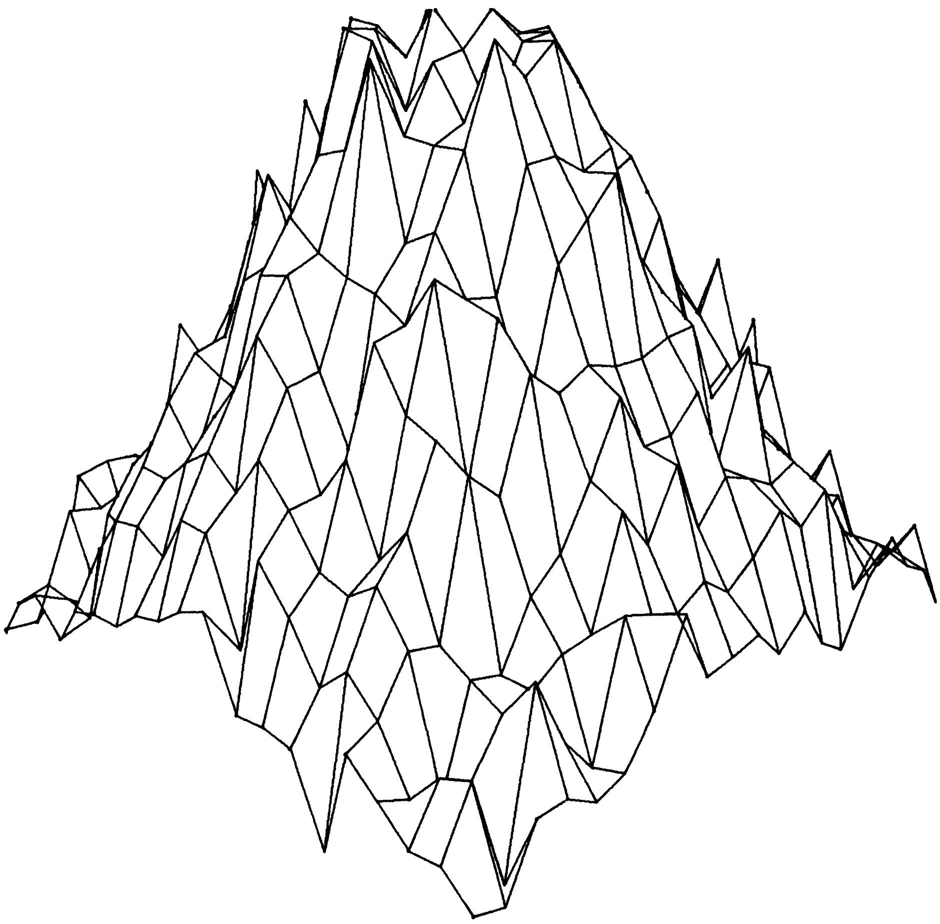


FIG. 2. *The data points for the example in* § 6.

It now follows from Lemma 4.2 that $\lim_{p\to\infty} \mu_{2,p}(\lambda_p^V) = 0$, and thus

$$\lim_{p\to\infty} E[T_p(\lambda_p^V)] = \lim_{p\to\infty} b_p^2(\lambda_p^V)\sigma^2\mu_{2,p}(\lambda_p^V) = 0,$$

and the proof is complete. □

The following theorem is the main result of the paper. It shows that λ_p^V is an asymptotically optimal estimate of the best smoothing parameter λ_p^T.

THEOREM 5.6. *Suppose that P_p is a sequence of partitions as in Lemma* 5.1. *Then*

$$\lim_{p\to\infty} \frac{E[T_p(\lambda_p^V)]}{E[T_p(\lambda_p^T)]} = 1. \tag{5.14}$$

Proof. By Lemma 4.1,

$$\begin{aligned} E[T_p(\lambda_p^V)](1-\Delta_p(\lambda_p^V)) &\leqq E[V_p(\lambda_p^V)] - \sigma^2 \\ &\leqq E[V_p(\lambda_p^T)] - \sigma^2 \\ &\leqq E[T_p(\lambda_p^T)](1+\Delta_p(\lambda_p^T)). \end{aligned}$$

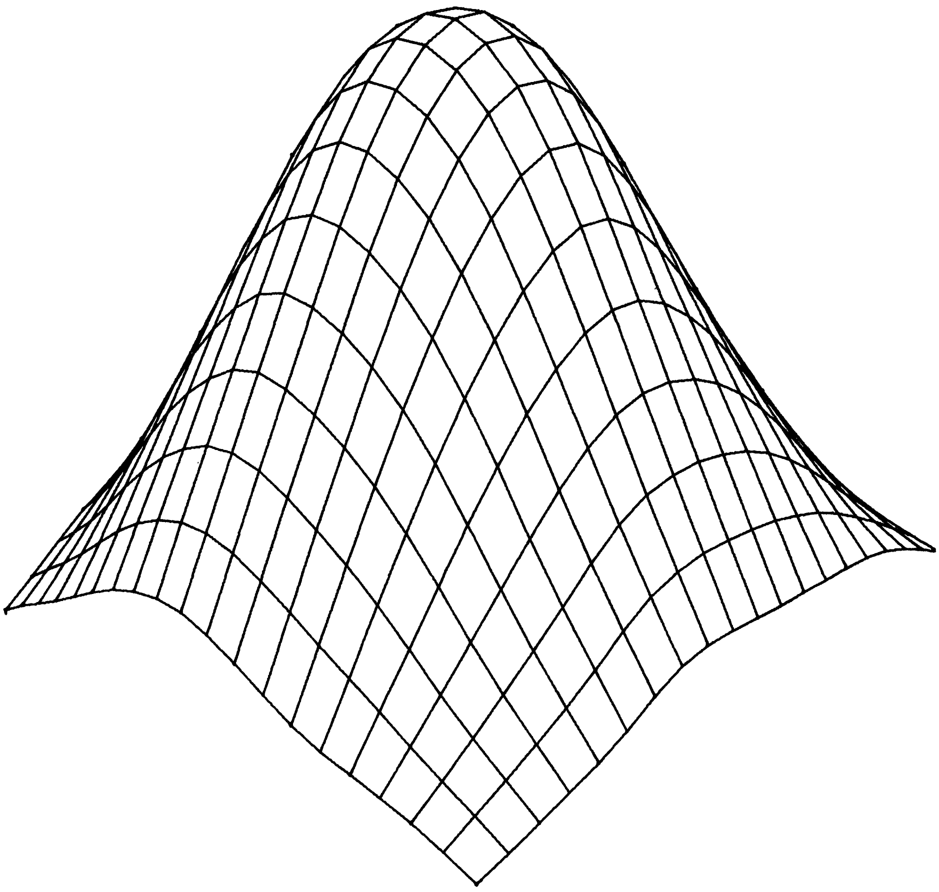


FIG. 3. *The smoothing spline fit for the example in* § 6.

It follows that

$$\frac{E[T_p(\lambda_p^V)]}{E[T_p(\lambda_p^T)]} \leq \frac{1+\Delta_p(\lambda_p^T)}{1-\Delta_p(\lambda_p^V)}.$$

Now by Lemmas 5.3, 5.4, and 5.5,

$$\lim_{p\to\infty} \Delta_p(\lambda_p^V) = \lim_{p\to\infty} \Delta_p(\lambda_p^T) = 0,$$

and the assertion of the theorem follows. □

6. A numerical example. In this section we illustrate the performance of the method on a typical example involving the bivariate function

$$f(x, y) = e^{-((x-0.6)/0.2)^2-((y-0.5)/0.2)^2}$$

on the unit square $\Omega = [0, 1]\times[0, 1]$. This function is illustrated in Fig. 1.

As data we take

$$z_{ij} = f(x_i, y_j) + \varepsilon_{ij},$$

where $x_i = ih$ and $y_i = ih$, $i = 0, \cdots, 20$, with $h = 1/20$, and where the errors are generated by a pseudorandom number generator. The errors are assumed to be normally distributed with mean zero and standard deviation 0.2. The measurement grid is shown in Fig. 2.

Fig. 3 shows the surface that results from fitting the above data with a tensor natural smoothing spline, where the smoothing parameter is selected by the generalized cross-validation method of this paper. The computations were carried out on an IBM PC/AT, and required approximately two minutes to complete.

7. Remarks. (1) It has been observed in practice, both in one and two variables, that the smoothing parameter λ_p^V produced by generalized cross validation performs almost as well as the optimal choice of the smoothing parameter, even for relatively modest numbers of data points.

(2) The fact that the smoothing spline provides an excellent fit of the function in the example in § 6 attests to the performance of the method. The fact that the example was computed on a PC in a rather short time attests to the efficiency and applicability of the method.

(3) It is a relatively simple matter to carry over the analysis of this paper to the cases of tensor complete and tensor periodic smoothing splines. For univariate results on complete smoothing, see [14], and for univariate results on periodic smoothing, see [23].

REFERENCES

[1] G. Baszenski and L. L. Schumaker, *Tensor products of abstract smoothing splines*, in Alfred Haar Memorial Conference, J. Szabados and K. Tandori, eds., North Holland, Amsterdam, the Netherlands, 1986, pp. 181-182.

[2] C. de Boor, *Efficient computer manipulation of tensor products*, ACM Trans. Math. Software, 5 (1979), pp. 173-182.

[3] 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.

[4] J. Duchon, *Interpolation des fonctions de deux variables suivant le principe de la flexion des plaques minces*, RAIRO Anal. Numér., 10 (1976), pp. 5-12.

[5] ———, *Sur l'erreur d'interpolation des fonctions de plusieurs variables par les D^m-splines*, RAIRO Anal. Numér., 12 (1978), pp. 325-334.

[6] ———, *Splines minimizing rotation-invariant semi-norms in Sobolev spaces*, in Multivariate Approximation Theory, W. Schempp & K. Zeller, eds., Birkhäuser Verlag, Basel, Switzerland, 1979, pp. 85-100.

[7] 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.

[8] C. Gu, D. M. Bates, Z. Chen, and G. Wahba, *The computation of* GCV *functions through Householder tridiagonalization with applications to the fitting of interactive spline models*, SIAM J. Matrix Anal., to appear.

[9] C. L. Hu and L. L. Schumaker, *Bivariate natural spline smoothing*, in Delay Equations, Approximation and Application, G. Meinardus and G. Nürnberger, eds., Birkhäuser Verlag, Basel, Switzerland, 1985, pp. 165-179.

[10] ———, *Complete spline smoothing*, Numer. Math., 49 (1986), pp. 1-10.

[11] M. F. Hutchinson and F. R. de Hoog, *Smoothing noisy data with spline functions*, Numer. Math., 50 (1987), pp. 311-319.

[12] T. Lyche and L. L. Schumaker, *Computation of smoothing and interpolating natural splines via local bases*, SIAM J. Numer. Anal., 10 (1973), pp. 1027-1038.

[13] M. Reed and B. Simon, *Functional Analysis*, Academic Press, New York, 1972.

[14] L. L. Schumaker and F. Utreras, *Asymptotic properties of complete smoothing splines and applications*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 24-38.

[15] L. Schwartz, *Théorie des Distributions*, Hermann, Paris, France, 1966.

[16] F. I. Utreras, *Cross validation techniques for smoothing spline functions in one or two dimensions*, in Smoothing Techniques for Curve Estimation, M. Rosenblatt and Th. Gasser, eds., Lecture Notes in Mathematics 757, Springer Verlag, Berlin, New York, 1979.

[17] F. I. UTRERAS, *Sur la choix du paramètre d'ajustement dans le lissage par fonctions spline*, Numer. Math., 34 (1980), pp. 15-28.

[18] ———, *Optimal smoothing of noisy data using spline functions*, SIAM J. Sci. Statist. Comput., 2 (1981), pp. 349-362.

[19] ———, *Natural spline functions, their associated eigenvalue problem*, Numer. Math., 42 (1983), pp. 107-117.

[20] ———, *Positive smoothing splines*, in Approximation Theory, V. C. Chui, L. Schumaker, and J. Ward, eds., Academic Press, New York, 1986, pp. 603-606.

[21] ———, *On generalized cross-validation for multivariate smoothing spline functions*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 430-443.

[22] ———, *Convergence rates for multivariate smoothing spline functions*, J. Approx. Theory, 52 (1988), pp. 1-27.

[23] G. WAHBA, *How to smooth curves and surfaces with splines and cross-validation*, in Proc. 24th Conference on Design of Experiments, U.S. Army Research Office, 1979, pp. 167-192.

[24] ———, *Spline bases, regularization, and generalized cross validation for solving approximation problems with large quantities of noisy data*, in Approximation Theory III, E. W. Cheney, ed., Academic Press, New York, 1980, pp. 905-912.

[25] ———, *Cross validated spline methods for the estimation of multivariate functions from data on functionals*, in Statistics, an Appraisal, H. A. D. and H. T. Davis, eds., Iowa State University Press, Ames, IA, 1984, pp. 205-235.

[26] G. WAHBA AND J. WENDELBERGER, *Some new mathematical methods for variational objective analysis using splines and cross validation*, Mon. Wea. Rev., 108 (1980), pp. 36-57.

[27] G. WAHBA AND S. WOLD, *A completely automatic French curve fitting spline function by cross validation*, Comm. Statist., 4 (1975), pp. 1-17.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 4, pp. 732-748, July 1990

CLOSED NEWTON-COTES QUADRATURE RULES FOR STIELTJES INTEGRALS AND NUMERICAL CONVOLUTION OF LIFE DISTRIBUTIONS*

MICHAEL TORTORELLA†

Abstract. A generalization of the Newton-Cotes quadrature rules that provides a means for numerical computation of Stieltjes integrals without using derivatives is described. The methods find wide application in the numerical evaluation of many applied probability models. Numerical convolution of life distributions is discussed in this paper. Error analyses are provided.

Key words. applied probability, numerical analysis

AMS(MOS) subject classifications. 65D30, 65U05, 60K30

1. Introduction. Many applied probability models require for their completion the evaluation of integrals of the form

$$(1) \qquad \int_{-\infty}^{\infty} F(t-u)\, dG(u),$$

where F and G are cumulative distribution functions (cdf). Equation (1) is the familiar convolution of F and G, and is often abbreviated to $F * G(t)$. If X and Y are independent random variables having cdf's F and G, respectively, then the distribution of their sum $X+Y$ is given by $F * G$.

Closed form expressions for the convolution (1) are available in only a few special cases. The closure of the normal and gamma families of cdf's under convolution is well known. The convolution of two distributions of phase type [1] is again a distribution of phase type. Other examples are available, such as a uniform distribution convolved with various other cdf's, but these often require a *tour de force* in synthetic integration. Furthermore, the evaluation of (1) may be taking place within a broader computer model. All these factors lead to an interest in evaluating convolutions of cdf's by numerical methods.

Cleroux and McConalogue [2] and McConalogue [3], [4] have previously studied this problem for life distributions. They rewrite (1) in the form

$$(2) \qquad \int_{0}^{t} F(t-u) g(u)\, du,$$

assuming that the density g of G exists and is accessible in a suitable form for numerical work. The interval of integration is $[0, t]$ because F and G are life distributions (i.e., have support in $[0, \infty[$). Their algorithm uses cubic spline interpolation of F and g over a subdivision of $[0, t]$ and a five-point Lobatto quadrature [5] on each panel to obtain the integral for each value of t separately. A requirement of the original algorithm [2] is that F and G have densities that are bounded on $[0, \infty[$, so that, for example, it fails for a decreasing failure rate Weibull distribution having shape parameter less than one. McConalogue [4] developed an improved algorithm that is able to tolerate certain singularities at the origin. However, this improved algorithm still cannot work with a decreasing failure rate Weibull distribution whose shape parameter is less than

* Received by the editors July 27, 1987; accepted for publication (in revised form) June 22, 1989.

† AT&T Bell Laboratories, Crawfords Corner Road, Holmdel, New Jersey 07733-1988.

one-half. Baxter [6] surveys other methods of numerical convolution and gives an example of the use of McConalogue's improved algorithm [4].

Shohat and Tamarkin [7] develop Gaussian quadrature rules for Stieltjes integrals. Schulten, Anderson, and Gordon [8] use Gaussian quadrature, with up to six nodes, for Stieltjes integrals to evaluate Airy integrals (in which the interval of integration is fixed). Gautschi [9] obtains quadrature rules based on polynomials orthogonal in L^2 of the measure determined by the Stieltjes integrator function. Two different approaches may be followed in applying Gaussian quadrature to numerical convolution of life distributions. First, one may construct a rule for each interval $[0, t]$ corresponding to each argument at which the value of $F * G(t)$ is desired. This is inconvenient because one is typically interested in the values of $F * G$ for many values of t, and in this implementation, the nodes of the rule must be recomputed for each new value of t. Also, evaluating the convolution at n points using a rule with m nodes requires $O(mn)$ function evaluations. To avoid this, one may use a single Gaussian rule on $[0, \infty[$ and incorporate the indicator function of the interval $[0, t]$ into the integrand. However, in this implementation, accuracy may suffer because of possible abrupt changes in the quadrature as t crosses the (fixed) nodes of the $[0, \infty[$ rule.

This paper documents closed Newton-Cotes quadrature rules [10] for Stieltjes integrals of the form

$$\int_a^b x(t)\, d\alpha(t), \tag{3}$$

and applies them to the develop a numerical convolution method for life distributions. Generally we will require, at least, x to be continuous and α to be of bounded variation over the interval $[a, b]$. If α were differentiable, and an expression for $D\alpha$ suitable for numerical work were accessible, we could write (3) as

$$\int_a^b x(t) D\alpha(t)\, dt, \tag{4}$$

and use one of many effective numerical quadrature rules to evaluate (4). Indeed, since we assume α is of bounded variation, it is differentiable almost everywhere on $[a, b]$. However, this fact is of little comfort for numerical calculations because the set on which α is differentiable may be awkward to account for in software. While these cases may be unusual in practice, we are aiming for a robust rule that makes this consideration unnecessary. If there were no expression for $D\alpha$ that could be put into a function call—for example, if α were only known tabularly—evaluating (4) would require numerical differentiation. Besides its potential for introducing instability problems, numerical differentiation within a numerical quadrature seems like extra effort that may be avoidable.

By treating the problem of numerical evaluation of Stieltjes integrals directly, we avoid numerical differentiation and provide a method that is simple and widely applicable. Explicit numerical analysis of this method is documented here. While the mathematics of this analysis is a straightforward generalization of the standard numerical analysis of the classical Newton-Cotes rules, the quadrature rules presented here provide a simple and robust solution to a significant problem in the evaluation of certain applied probability models. We have also used these rules to implement numerical solution of integral equations $r(t) = h(t) + \int_0^t r(t-u)\, dF(u)$ of renewal type [11], details of which will be published subsequently.

The remainder of this paper is organized as follows. Section 2 contains the derivation of Newton-Cotes quadrature rules for integrals of the form (3). We give explicit expressions for the rules that correspond to the trapezoidal and Simpson rules in the ordinary ($\alpha(t)=t$) case. Section 3 discusses numerical analysis for the closed Newton-Cotes rules. Section 4 covers the implementation of trapezoidal and Simpson rules in the case of convolution of life distributions. We conclude with some examples that illustrate the accuracy of the two rules in cases in which the convolution function is known exactly.

2. Newton–Cotes quadrature rules for Stieltjes integrals.

2.1. A simple, but incomplete, approach. Because the integral (3) is taken with respect to the measure μ_α, $\mu_\alpha(E) := \int_E d\alpha(t)$, one might expect to obtain quadrature rules for this integral from quadrature rules for "ordinary" ($\alpha(t)=t$) Riemann integrals by using first differences $\alpha(b)-\alpha(a)$ of α in place of the "ordinary" increment $b-a$. For instance, setting $m=(a+b)/2$, the ordinary Simpson rule is

$$\int_a^b x(t)\,dt \approx \frac{b-a}{6}[x(a)+4x(m)+x(b)]. \tag{5}$$

A simple thing to do here would be to replace $b-a$ by $\alpha(b)-\alpha(a)$, obtaining

$$\int_a^b x(t)\,d\alpha(t) \approx \frac{\alpha(b)-\alpha(a)}{6}[x(a)+4x(m)+x(b)]. \tag{6}$$

This is a step in the right direction, but it will be inaccurate for rules of degree greater than that of the trapezoidal rule because it ignores the variation of α over the subintervals. In the example, we would want to involve both $\alpha(m)-\alpha(a)$ and $\alpha(b)-\alpha(m)$ to increase accuracy. In the general case, this consideration leads to the expressions in the next section.

2.2. The Newton–Cotes rules. When these "interior" differences of α are taken into account, higher-order differences of α come into play for the Simpson and all higher-degree rules. Nonetheless, even if α were known only in tabular form, the rules are simple both conceptually and computationally. In this section, we will derive the Newton-Cotes rules for the Stieltjes integral (3).

2.2.1. Notation and preliminaries. Let X be a normed linear space of continuous functions on $[a, b]$ (typically, X will consist of functions that are differentiable a certain number of times), and let J be the integration functional on X, $Jx := \int_a^b x(t)\,dt$, $x \in X$. Let A be a subspace of X, and let P be a projection of X onto A. Let Q be a quadrature rule for J. If the quadrature rule is constructed by integrating the images in A of the projection P of certain functions in X, then it will be called *projection-based.* Note that A consists of functions for which the quadrature rule is exact, although the rule may be exact for other functions as well.

As a final point of notation, we will let $J_\alpha x$ stand for $\int_a^b x(t)\,d\alpha(t)$, where $x \in X$ and α is of bounded variation on $[a, b]$.

2.2.2. Review of the Newton–Cotes rules for ordinary Riemann integrals. The Newton-Cotes rules for ordinary Riemann integrals are projection-based. To obtain the closed Newton-Cotes rule of degree n, interpolate x with a polynomial of degree n at $n+1$ equally spaced points, including the endpoints, in $[a, b]$, and then explicitly integrate the interpolation polynomial. Formally, choose a subdivision of $[a, b]$ containing $n+1$ points

$$a = t_0 < \cdots < t_n = b \tag{7}$$

that divide the interval into n subintervals of equal length. Let A_n be the subspace of X consisting of the polynomials of degree at most n, and let P_n be the projection of X onto A_n defined by P_nx is the polynomial of least degree not exceeding n that interpolates x at $t_0, \cdots, t_n$. The closed Newton-Cotes rule of degree n for Jx is then simply $Q_nx := JP_nx$. The closed Newton-Cotes rule of degree n is exact for polynomials of degree n when n is odd, and is exact for polynomials of degree $n+1$ when n is even [12]. Explicit expressions for the Newton-Cotes rules up to and including degree ten, including error estimates assuming suitable differentiability conditions on x, are given in § 25.4 of Abramowitz and Stegun [5].

The open Newton-Cotes rule of degree n is obtained in exactly the same fashion, except that the subdivision contains $n+3$ equally spaced points, and the endpoints $a = t_0$ and $b = t_{n+2}$ are not used. In this case, an nth degree polynomial is used that interpolates x at $t_1, \cdots, t_{n+1}$. We will only study closed Newton-Cotes rules for Stieltjes integrals, so other properties of the open Newton-Cotes rules for ordinary Riemann integrals will not be reviewed here.

2.3. The closed Newton-Cotes rules for Stieltjes integrals.

2.3.1. Definition. The closed Newton-Cotes rules for Stieltjes integrals are projection-based also. We will continue to use the subdivision (7), the subspace A_n, and the projection P_n, as before. However, we will now impose the condition that α be continuous, as well as of bounded variation, on $[a, b]$. Remembering that one of our goals is to avoid differentiations of either x or α, we proceed as follows for the Newton-Cotes rule of degree n. Begin, as in the ordinary case, by approximating the integral $J_\alpha x$ by $J_\alpha(P_nx)$. Then integrate by parts to obtain $x(b)\alpha(b) - x(a)\alpha(a) - J(\alpha D(P_nx))$, recalling that $P_nx(a) = x(a)$ and $P_nx(b) = x(b)$. This is not the final step, though, because $\alpha D(P_nx)$ may not be integrable in closed form. Therefore, we apply the projection again, and obtain the closed Newton-Cotes quadrature rule of degree n for Stieltjes integrals as

$$Q_n^S(x, \alpha) := x(b)\alpha(b) - x(a)\alpha(a) - JP_n(\alpha D(P_nx)). \tag{8}$$

We will now show that (8) reduces to the classical Newton-Cotes rule in the ordinary case, that is, when $\alpha(t) = t$.

THEOREM 1. *Let $\alpha_0(t) = t$. Then for every x for which Q_nx exists, $Q_n^S(x, \alpha_0) = Q_nx$.*

Proof. Since Q_nx exists, we have $P_nx \in A_n$. Then $D(P_nx) \in A_{n-1}$, and $\alpha_0 D(P_nx) \in A_n$. Since P_n is a projection on A_n, we obtain $P_n(\alpha_0 D(P_nx)) = \alpha_0 D(P_nx)$. It follows that the integral term in (8) reduces to $J(\alpha_0 D(P_nx))$, and integrating this by parts produces the result. □

It is natural to ask at this point if $Q_n^S(x, \alpha)$ reduces to $Q_n(xD\alpha)$ when α is differentiable. The answer is *no*, in general; to see this, take $\beta(t) = t^2$ and compute $Q_1^S(x, \beta) \neq Q_1(xD\beta)$. One consequence of this is that error analysis for (8) is not just a simple matter of applying the standard error analysis for Q_n to $xD\alpha$. This is discussed further in § 3.1.

2.3.2. The trapezoidal and Simpson rules. In this section, we will give explicit expressions for Q_1^S, the trapezoidal rule, and Q_2^S, the Simpson's rule, for Stieltjes integrals. Other than noting that we use Newton's form of the interpolating polynomial, with coefficients given as divided differences [12],

$$P_ny(t) = y[t_0] + \sum_{i=1}^{n} y[t_0, \cdots, t_i]\omega_{i-1}(t), \tag{9}$$

where

$$\omega_i(t) = (t - t_0) \cdots (t - t_i), \qquad i = 0(1)n, \tag{10}$$

we will omit details of algebraic manipulations. Using (9) for $n=1$ and $n=2$ and inserting into (8), we obtain

$$Q_1^S(x,\alpha):=\frac{x(b)+x(a)}{2}[\alpha(b)-\alpha(a)], \tag{11}$$

and

$$Q_2^S(x,\alpha):=\frac{\alpha(b)-\alpha(a)}{6}[x(a)+4x(m)+x(b)] \\ +\frac{x(b)-x(a)}{3}[\alpha(a)-2\alpha(m)+\alpha(b)], \tag{12}$$

where $m=(a+b)/2$. While the trapezoidal rule Q_1^S looks like the simple one you would get by following the prescription of § 2.1, only the first term of the Simpson's rule Q_2^S looks like this. The second term, involving a second difference of α, is a correction that takes into account the variation of α over $[a, m]$ and $[m, b]$.

The next section discusses error analysis for the closed Newton-Cotes quadrature rules for Stieltjes integrals. Section 4 shows how the Simpson's rule is used to provide a simple, robust, and accurate quadrature for the convolution of life distributions.

3. Numerical analysis.

3.1. Error analysis.

3.1.1. Introduction. The goal of this section is to express the error in the quadrature rule (8) in terms of properties of x and α. Accordingly, we define $E_n^S(x,\alpha):=J_\alpha x-Q_n^S(x,\alpha)$. By an integration by parts, we obtain $E_n^S(x,\alpha)=J(P_n(\alpha DP_nx)-\alpha Dx)$. Adding and subtracting αDP_nx, and using the linearity of J and one more integration by parts, we finally obtain

$$E_n^S(x,\alpha)=J_\alpha((I-P_n)x)-J((I-P_n)(\alpha DP_nx)), \tag{13}$$

where I is the identity operator, $Ix=x$. Equation (13) forms the basis for the error analyses presented in this section.

3.1.2. Results. As in the case of the ordinary Newton-Cotes rules, it is convenient to separate the two cases, n even and n odd. The results given in this section show that the error in the quadrature rule (8) is controlled by intermediate values of certain derivatives of the functions x and α.

THEOREM 2. *Let the points of* (7) *divide* $[a, b]$ *into an even number of subintervals of equal length. Let* x *and* α *have continuous derivatives of order* $n+2$ *on* $[a, b]$. *Then there are* τ_1 *and* τ_2 *strictly between* a *and* b, *and a constant* $K_n<0$, *for which*

$$E_n^S(x,\alpha)=\frac{K_n}{(n+2)!}\Bigg[D^{n+2}x(\tau_1)D\alpha(\tau_1)+(n+2)D^{n+1}x(\tau_1)D^2\alpha(\tau_1) \\ -\sum_{k=3}^{n+2}\binom{n+2}{k}D^k\alpha(\tau_2)D^{n+3-k}P_nx(\tau_2)\Bigg]. \tag{14}$$

Proof. Let $\omega_n(t)=(t-t_0)\cdots(t-t_n)$. The starting point of the proof is (13) and the equality $(I-P_n)y(t)=\omega_n(t)y[t_0,\cdots,t_n,t]$ (see Equation 6.1.8 of Isaacson and

Keller [12]). Writing $\Omega_n(t)=\int_a^t \omega_n(s)\,ds$, the first term in (13) becomes

$$\begin{aligned} J_\alpha((I-P_n)x) &= \int_a^b \omega_n(t)x[t_0,\cdots,t_n,t]\,d\alpha(t) \\ &= \int_a^b D\Omega_n(t)x[t_0,\cdots,t_n,t]D\alpha(t)\,dt \\ &= -\int_a^b \Omega_n(t)D\{x[t_0,\cdots,t_n,\cdot\,]D\alpha\}(t)\,dt. \end{aligned}$$

The first term in the integration by parts vanishes by Lemma 7.1.4 of [12]. Continuing,

$$\begin{aligned} J_\alpha((I-P_n)x) &= \int_a^b [D\{x[t_0,\cdots,t_n,\cdot\,]\}(t)D\alpha(t) \\ &\qquad + x[t_0,\cdots,t_n,t]D^2\alpha(t)]\Omega_n(t)\,dt \\ &= -\int_a^b \left[\frac{1}{(n+2)!}D^{n+2}x(\tau_1(t))D\alpha(t)\right. \\ &\qquad \left. + \frac{1}{(n+1)!}D^{n+1}x(\tau_2(t))D^2\alpha(t)\right]\Omega_n(t)\,dt \\ &= K_n\left[\frac{1}{(n+2)!}D^{n+2}x(\tau_1)D\alpha(\tau_1)+\frac{1}{(n+1)!}D^{n+1}x(\tau_1)D^2\alpha(\tau_1)\right]. \end{aligned}$$

We have used Problem 6.1.6 and Corollary 6.1.2.2 of [12].

For the second term in (13), we have $J((I-P_n)(\alpha DP_nx))=\int_a^b \omega_n(t)\{\alpha DP_nx\}[t_0,\cdots,t_n,t]\,dt$. Here we can apply Theorem 7.1.1 of [12] directly, obtaining

$$\begin{aligned} J((I-P_n)(\alpha DP_nx)) &= \frac{K_n}{(n+2)!}D^{n+2}(\alpha DP_nx)(\tau_2) \\ &= \frac{K_n}{(n+2)!}\sum_{k=0}^{n+2}\binom{n+2}{k}D^k\alpha(\tau_2)D^{n+3-k}P_nx(\tau_2) \\ &= \frac{K_n}{(n+2)!}\sum_{k=3}^{n+2}\binom{n+2}{k}D^k\alpha(\tau_2)D^{n+3-k}P_nx(\tau_2), \end{aligned}$$

the last equality following because *degree*$(P_nx)\leqq n$. □

THEOREM 3. *Let the points of* (7) *divide* $[a,b]$ *into an odd number of subintervals of equal length. Let* x *and* α *have continuous derivatives of order* $n+1$ *on* $[a,b]$. *Then there exist* τ_1, τ_2, τ_3, τ_4, *and* τ_5 *strictly between* a *and* b, *and constants* $K_n^1\geqq 0$ *and* $K_n^2<0$ *for which*

$$\begin{aligned} E_n^S(x,\alpha) &= \frac{K_n^1}{n!}[D^nx(\tau_1)D^2\alpha(\tau_2)-D^nx(\tau_3)D^2\alpha(\tau_3)] \\ &\quad + \frac{K_n^2}{(n+1)!}\left[D^{n+1}x(\tau_4)D\alpha(\tau_4)+\sum_{k=2}^{n+1}\binom{n+1}{k}D^k\alpha(\tau_5)D^{n+2-k}P_nx(\tau_5)\right]. \end{aligned} \tag{15}$$

Proof. Write the first term in the error expression (13) as

$$\begin{aligned} J_\alpha((I-P_n)x) &= \int_a^{b-h}\omega_n(t)x[t_0,\cdots,t_n,t]D\alpha(t)\,dt \\ &\quad + \int_{b-h}^b \omega_n(t)x[t_0,\cdots,t_n,t]D\alpha(t)\,dt. \end{aligned}$$

By an application of the mean value theorem for integrals, the second integral becomes

$$\frac{1}{(n+1)!} D^{n+1}x(\tau_1)D\alpha(\tau_1) \int_{b-h}^{b} \omega_n(t)\,dt.$$

For the first integral, write $\omega_n(t)=(t-t_n)\omega_{n-1}(t)$ and $\Omega_{n-1}(t)=\int_a^t \omega_{n-1}(s)\,ds$, and use the basic property of divided differences (equation 6.1.6 of [12]) to obtain

$$\int_a^{b-h} D\Omega_{n-1}(t)D\alpha(t)x[t_0,\cdots,t_{n-1},t]\,dt - x[t_0,\cdots,t_n]\int_a^{b-h} D\Omega_{n-1}(t)D\alpha(t)\,dt.$$

Integrate by parts in each term, noting that the integrated terms vanish, as in the proof of Theorem 2, above, because $n-1$ is even. Then use the mean value theorems for integrals and for divided differences as we did in the proof of Theorem 2, obtaining for the first integral

$$\int_a^{b-h} \Omega_{n-1}(t)\,dt\left[\frac{1}{n!}D^n x(\tau_2)D\alpha(\tau_2)+\frac{1}{(n+1)!}D^{n+1}x(\tau_3)D\alpha(\tau_3)\right].$$

Combining the first and second integrals, and combining the constants using an integration by parts, and renumbering the indices on the τ's, we obtain

$$\frac{K_n^1}{n!}[D^n x(\tau_1)D^2\alpha(\tau_2)-D^n x(\tau_3)D^2\alpha(\tau_3)]+\frac{K_n^2}{(n+1)!}D^{n+1}x(\tau_4)D\alpha(\tau_4)$$

for the first term in the error (13). For the second term in the error (13), we obtain, as in the proof of Theorem 2,

$$\frac{K_n^2}{(n+1)!}\sum_{k=2}^{n+1}\binom{n+1}{k}D^k\alpha(\tau_5)D^{n+2-k}P_n x(\tau_5). \qquad \square$$

3.1.3. Discussion. This section will contain some observations on the numerical analysis of the Newton-Cotes rules for Stieltjes integrals as embodied in the two theorems above.

When $\alpha(t)=t$, the second term in the error (13) vanishes (see the proof of Theorem 1), and the error analysis then applied to the first term produces the familiar error expressions as found, for example, in [5]. This can also be seen by putting $\alpha(t)=t$ in (14) and (15).

If x and α are polynomials, (14) and (15) show that the quadrature rule (8) is exact as long as $degree(x)+degree(\alpha)\leqq n+2$ if n is even, and $degree(x)+degree(\alpha)\leqq n+1$ if n is odd. Thus the rule (8) has the same degree of precision as that of $Q_n(xD\alpha)$, when α is a polynomial, even though the appearance of the formulas for (8) and for $Q_n(xD\alpha)$ is different.

The error analysis as embodied in Theorems 2 and 3 is perhaps not very useful as it stands so far, because no prescription for determining the τ's is available. Common practice in the ordinary case is to bound the derivatives appearing in (14) and (15), thereby producing a bound on E_n. We will follow this practice here. However, because of the integration by parts that led to (8), we also have to bound the magnitude of a term of the form D^rP_nx for some r. There are (at least) two approaches to this task. One approach is first to use Markov's inequality (§ 3.3 of [13]) to establish that $\|D|A_n\|=2n^2/(b-a)$. From this it follows that $\|D^rP_nx\|_\infty\leqq(2n^2/(b-a))^r\|P_n\|\|x\|_\infty$. The norms of the interpolation operators P_n are given in § 4.4 of Powell [14]. Table 4.5 of [14] shows how rapidly the norms of P_n grow with n when, for example, $a=-5$, $b=5$. Another approach is to explicitly evaluate the derivatives, using (9). The error bounds (16) and (17), below, are examples of the use of the former approach. We use the latter approach in § 4.2, which details an error analysis for numerical convolution of life distributions using the trapezoidal and Simpson rules.

Let y be a function that has a continuous derivative of order r on $[a, b]$. Let $B_r(y)$ denote $\sup\{|D^r y(t)|: a \leqq t \leqq b\}$. In particular, $B_0(y) = \|y\|_\infty$. Then, from Theorem 2, it follows that for n even,

$$(16)\quad |E_n^S(x, \alpha)| \leqq \frac{K_n}{(n+2)!}\Bigg[B_{n+2}(x)B_1(\alpha) + (n+2)B_{n+1}(x)B_2(\alpha) + B_0(x)\|P_n\| \sum_{k=3}^{n+2} \binom{n+2}{k}\left(\frac{2n^2}{b-a}\right)^{n+3-k} B_k(\alpha)\Bigg],$$

and from Theorem 3 it follows that, for n odd,

$$(17)\quad |E_n^S(x, \alpha)| \leqq \frac{2K_n^1}{n!}[B_n(x)B_2(\alpha)] + \frac{K_n^2}{(n+1)!}\Bigg[B_{n+1}(x)B_1(\alpha) + B_0(x)\|P_n\| \sum_{k=2}^{n+1} \binom{n+1}{k}\left(\frac{2n^2}{b-a}\right)^{n+2-k} B_k(\alpha)\Bigg].$$

Expressions for the constants are given in Table 1.

It can readily be seen that, because of the rapid growth of $\|D|A_n\|$ and $\|P_n\|$ with n, the smaller values of n are likely to yield rules that are more effective in practice. Consequently, if the interval of integration is large, it is probably wiser to use a compound rule with a smaller n that it is to use a single rule with large n. Compound rules are discussed in § 3.2.

TABLE 1
Constants for (14), (15), (16), *and* (17).

n	$\frac{K_n}{(n+2)!}$	$\frac{K_n^1}{n!}$	$\frac{K_n^2}{(n+1)!}$
1		0	$-h^3/12$
2	$-h^5/90$		
3		$2h^5/45$	$-3h^5/80$
4	$-8h^7/945$		
5		$16h^7/315$	$-275h^7/12096$
6	$-9h^9/1400$		
7		$9h^9/175$	$-8183h^9/518400$
8	$-2368h^{11}/467775$		
9		$4736h^{11}/93555$	$-173h^{11}/14620$
10	$-673175h^{13}/163459296$		

It is somewhat unsatisfying that the error estimates (16) and (17) involve higher derivatives of α, especially since what was sought was a rule that could be used in the absence of any differentiability properties of α. In particular, this means that while the convolution algorithms (25) and (26(a)-(b)), below, do work with continuous cdf's that may lack everywhere-defined densities, such as the Weibull distribution with shape parameter less than one, it will not be possible to provide an error analysis for these cdf's using this technique. Recall, though, that the conventional error estimates for the ordinary Newton-Cotes rules involve higher derivatives of the integrand. This is a feature of the method used, which seeks the maximum order that the rule can possibly achieve. Fewer differentiability assumptions lead to the same rules having lower order. Should α be known only in tabular form, the error estimates can be viewed as the error that could be achieved if one were to interpolate the α table with a sufficiently differentiable function. Error estimates under decreased differentiability assumptions would be useful in case one had an α that was known to have a certain number of continuous derivatives, and no more (for example, if α were a cubic spline, its third derivative could be discontinuous at the knots). This remains a fruitful area for further research whose ultimate goal would be an error bound for (8) when x is merely continuous, and α is merely of bounded variation. We given an error bound for the compound trapezoidal rule under similar conditions in § 3.3, below.

3.2. Compound rules. Suppose one has a large interval of integration, say, $[a, z]$. At least two courses of action are possible. One is to take a subdivision of the form (7), with z in place of b, and use a rule of high degree based on those points. This will work well if x and α are smooth (higher derivatives are small). However, this procedure is not likely to be robust against wiggly integrands or integrators. Another choice is to take a subdivision like (7) again, but now use a low degree rule on each subinterval (or group of subintervals) separately. This yields a compound rule, familiar from the ordinary case. Suffice it to say that the order of a compound rule, using subintervals of equal length, is one less than the order of the simple rule from which it is built, a result familiar from the ordinary case that carries over to the Stieltjes case. However, there is no need for the subintervals in a compound rule to have equal length. This permits continued use of adaptive schemes that put more subdivision points in areas where the functions change more rapidly, and fewer points where the functions are smoother.

3.3. Compound trapezoidal rule with α of bounded variation. The compound trapezoidal rule on a subdivision $a=t_0<t_1<\cdots<t_N=z$ (not necessarily equally spaced) is

$$(18)\qquad Q_1^S(x, \alpha; N) := \sum_{i=1}^{N} \frac{x(t_{i-1})+x(t_i)}{2}[\alpha(t_i)-\alpha(t_{i-1})].$$

As a first step toward deriving error estimates for the quadrature rules (8) under more general conditions, we show what can be achieved with the compound trapezoidal rule with equally spaced points when α is a function of bounded variation.

Choose a subdivision of equally spaced points $a=t_0<t_1<\cdots<t_N=z$ with $h:=(z-a)/N=t_k-t_{k-1}$, $k=1(1)N$. Recall that a function f on an interval E is said to be Hölder continuous with exponent p $(0<p\leqq 1)$ if there is a positive number L such that $|f(s_1)-f(s_2)|\leqq L|s_1-s_2|^p$ for every $s_1, s_2\in E$. We then have the following result.

THEOREM 4. *If x is a Hölder continuous function of exponent p on $[a, z]$ and α is of bounded variation on $[a, z]$, then*

$$(19)\qquad \left|\int_a^z x(t)\,d\alpha(t)-Q_1^S(x, \alpha; N)\right|\leqq L\left(\frac{h}{2}\right)^p V_a^z(\alpha),$$

where $V_a^z(\alpha)$ is the total variation of α over $[a, z]$.

Proof. First we prove this for α a bounded nondecreasing step (piecewise constant) function on $[a, z]$, continuous from the right. Accordingly, we assume that there is a finite number of points $T_1, \cdots, T_J$ in $]a, z[$ at wich α is discontinuous but continuous from the right, $\alpha(T_i)-\alpha(T_i^-)=\sigma_i>0$, $i=1(1)J$, and α is constant between jumps. Let

$A_k = \{i\colon T_i \in]t_{k-1}, t_k]\}$, $k = 1(1)N$; note that $\sum_{i\in A_k} \sigma_i = \alpha(t_k) - \alpha(t_{k-1})$. If A_k is empty, then $\alpha(t_k) - \alpha(t_{k-1}) = 0$. If A_k is nonempty, then there is a $\tau_k \in]t_{k-1}, t_k]$ for which $\sum_{i\in A_k} \sigma_i x(T_i) = x(\tau_k) \sum_{i\in A_k} \sigma_i$ by the intermediate value theorem. Then

$$\begin{aligned}
&\left|\int_a^z x(t)\,d\alpha(t) - Q_1^S(x,\alpha;N)\right| \\
&\quad = \left|\sum_{k=1}^N \sum_{i\in A_k} \sigma_i x(T_i) - \sum_{k=1}^N \frac{x(t_{k-1})+x(t_k)}{2}[\alpha(t_k)-\alpha(t_{k-1})]\right| \\
&\quad = \left|\sum_{k=1}^N x(\tau_k) \sum_{i\in A_k} \sigma_i - \sum_{k=1}^N \frac{x(t_{k-1})+x(t_k)}{2}[\alpha(t_k)-\alpha(t_{k-1})]\right| \\
&\quad \leqq \sum_{k=1}^N \left|\frac{x(t_k)+x(t_{k-1})}{2} - x(\tau_k)\right| [\alpha(t_k)-\alpha(t_{k-1})] \\
&\quad \leqq \frac{L}{2}\sum_{k=1}^N [(\tau_k - t_{k-1})^p + (t_k-\tau_k)^p][\alpha(t_k)-\alpha(t_{k-1})] \\
&\quad \leqq L(h/2)^p[\alpha(b)-\alpha(a)] = L(h/2)^p V_a^z(\alpha)
\end{aligned}$$

since the maximum of $(v-s)^p + (s-u)^p$ for $s \in [u, v]$, $v - u = h$, is $2^{1-p}h^p$.

Now suppose α is a bounded nondecreasing function on $[a, z]$. Choose $\varepsilon > 0$. Then there is a step function α_ε, continuous from the right, for which $V_a^z(\alpha - \alpha_\varepsilon) < \varepsilon$ and

$$\left|\int_a^z x(t)\,d\alpha(t) - \int_a^z x(t)\,d\alpha_\varepsilon(t)\right| < \varepsilon.$$

Note that $|V_a^z(\alpha) - V_a^z(\alpha_\varepsilon)| < \varepsilon$ as well. Then

$$\begin{aligned}
&\left|\int_a^z x(t)\,d\alpha(t) - Q_1^S(x,\alpha;N)\right| \\
&\quad \leqq \left|\int_a^z x(t)\,d\alpha(t) - \int_a^z x(t)\,d\alpha_\varepsilon(t)\right| \\
&\quad\quad + \left|\int_a^z x(t)\,d\alpha_\varepsilon(t) - Q_1^S(x,\alpha_\varepsilon;N)\right| + |Q_1^S(x,\alpha_\varepsilon;N) - Q_1^S(x,\alpha;N)| \\
&\quad \leqq \varepsilon + L\left(\frac{h}{2}\right)^p V_a^z(\alpha_\varepsilon) + \sum_{k=1}^N \left|\frac{x(t_{k-1})+x(t_k)}{2}\right| \\
&\quad\quad \cdot |\alpha(t_k) - \alpha_\varepsilon(t_k) - (\alpha(t_{k-1}) - \alpha_\varepsilon(t_{k-1}))|.
\end{aligned}$$

The last term is bounded above by $B_0(x)V_a^z(\alpha - \alpha_\varepsilon) < \varepsilon B_0(x)$, so altogether we have

$$\left|\int_a^z x(t)\,d\alpha(t) - Q_1^S(x,\alpha;N)\right| < \varepsilon + L\left(\frac{h}{2}\right)^p V_a^z(\alpha) + \varepsilon L\left(\frac{h}{2}\right)^p + \varepsilon B_0(x).$$

Since ε is arbitrary, this establishes (19) for α bounded and nondecreasing.

For the final step, we assume α is of bounded variation on $[a, z]$. Then $\alpha = \alpha_1 - \alpha_2$ with α_1 and α_2 bounded and nondecreasing on $[a, z]$. Since $Q_1^S(x, \alpha; N)$ is linear in

α, we have by the definition of the Stieltjes integral for α of bounded variation

$$\left|\int_a^z x(t)\,d\alpha(t) - Q_1^S(x,\alpha;N)\right| \leqq \sum_{i=1}^{2}\left|\int_a^z x(t)\,d\alpha_i(t) - Q_1^S(x,\alpha_i;N)\right|$$
$$\leqq L\left(\frac{h}{2}\right)^p \sum_{i=1}^{2} V_a^z(\alpha_i) = L\left(\frac{h}{2}\right)^p V_a^z(\alpha).$$

This completes the proof. □

3.4. Romberg-like integration. We can also derive a Romberg-like procedure for increasing the precision of the compound trapezoidal rule. To do this, we need to assume that α is bounded and nondecreasing (this is no loss of generality because a function of bounded variation can be written as the difference of two bounded nondecreasing functions), and that we choose a subdivision satisfying $h=\alpha(t_i)-\alpha(t_{i-1})$ for $i=1(1)N$. That is, the subdivision points are not equally spaced, but are arranged so that when plotting α against t there is equal vertical spacing on the α axis. The following result permits the construction of a Romberg table [15] for the compound trapezoidal rule (18) in a similar way as for the ordinary trapezoidal rule.

THEOREM 5. *Suppose α is a bounded nondecreasing function on $[a, z]$, x and α have continuous derivatives of order 2ν on $[a, z]$, and h is as above. Then*

$$\int_a^z x(t)\,d\alpha(t) = Q_1^S(x,\alpha;N) - \sum_{j=1}^{\nu-1} c_j h^{2j} + O(h^{2\nu+1}), \tag{20}$$

where c_j is independent of h.

Proof. We apply Euler's expansion for the remainder in the ordinary trapezoidal rule, (11.3.13) of [10], to

$$\int_{t_{k-1}}^{t_k} x(t)\,d\alpha(t) = \int_{\alpha(t_{k-1})}^{\alpha(t_k)} x(\alpha^{-1}(u))\,du,$$

obtaining

$$\frac{x(t_{k-1})+x(t_k)}{2}[\alpha(t_k)-\alpha(t_{k-1})] - \sum_{j=1}^{\nu-1}\frac{[\alpha(t_k)-\alpha(t_{k-1})]^{2j}}{(2j)!}\beta_{2j}[D^{2j-1}(x\circ\alpha^{-1})]_{\alpha(t_{k-1})}^{\alpha(t_k)} \tag{21}$$
$$+O([\alpha(t_k)-\alpha(t_{k-1})]^{2\nu+1}),$$

where β_i is the ith Bernoulli number (§ 1.1 of [10]). Summing (21) over the subdivision and noting that $\alpha(t_k)-\alpha(t_{k-1})=h$, $k=1(1)N$, we obtain

$$\int_a^z x(t)\,d\alpha(t) = Q_1^S(x,\alpha;N) - \sum_{j=1}^{\nu-1}\frac{h^{2j}\beta_{2j}}{(2j)!}\sum_{k=1}^{N}[D^{2j-1}(x\circ\alpha^{-1})]_{\alpha(t_{k-1})}^{\alpha(k)} + O(h^{2\nu+1})$$
$$= Q_1^S(x,\alpha;N) - \sum_{j=1}^{\nu-1}\frac{h^{2j}\beta_{2j}}{(2j)!}[D^{2j-1}(x\circ\alpha^{-1})]_{\alpha(a)}^{\alpha(b)} + O(h^{2\nu+1}),$$

which establishes (20).

Put $t_k^{(1)} := t_k$, $k=1(1)N$ and $Q_{1,1}^S(x,\alpha;N) := Q_1^S(x,\alpha;N)$. To complete the Romberg table, choose a subdivision containing $2N+1$ points $a=t_0^{(2)}<t_1^{(2)}<\cdots<t_{2N}^{(2)}=b$ such that $\alpha(t_k^{(2)})-\alpha(t_{k-1}^{(2)})=h/2$, $k=1(1)2N$. Then set $Q_{1,2}^S(x,\alpha;2N) := (\frac{4}{3})Q_{1,1}^S(x,\alpha;2N)-(\frac{1}{3})Q_{1,1}^S(x,\alpha;N)$. By (20) we have

$$\int_a^b x(t)\,d\alpha(t) = Q_{1,2}^S(x,\alpha;2N) - \frac{1}{3}\sum_{j=2}^{\nu-1}\left(1-\frac{4}{2^{2j}}\right)c_j h^{2j} + O(h^{2\nu+1}), \tag{22}$$

showing that the error is now of order h^4. In general, continue with a subdivision $a=t_0^{(r)}<t_1^{(r)}<\cdots<t_{2^rN}^{(r)}=b$ with $\alpha(t_k^{(r)})-\alpha(t_{k-1}^{(r)})=h/2^r$, $r=0,1,2,\cdots$, and set

$Q^S_{1,r}(x, \alpha; 2^rN) := (4^r-1)^{-1}[4^rQ^S_{1,r-1}(x, \alpha; 2^rN) - Q^S_{1,r-1}(x, \alpha; 2^{r-1}N)]$. Repeated use of (20) shows that

$$\int_a^z x(t)\, d\alpha(t) - Q^S_{1,r}(x, \alpha; 2^rN) = O(h^{2r})$$

as long as $r \leqq \nu - 1$. $\square$

4. Numerical convolution of life distributions. In this section, we show how to use the trapezoidal and Simpson rules for Stieltjes integrals to construct simple and effective methods for numerical convolution of life distributions.

4.1. Using the trapezoidal and Simpson rules. Let F and G be life distributions, i.e., cumulative distribution functions, continuous from the right, with $F(0) = G(0) = 0$. We will study the Stieltjes convolution

$$F * G(t) = \int_0^t F(t-u)\, dG(u). \tag{23}$$

We assume that F and G are piecewise continuous. The quadrature rules below must be applied to each interval of continuity separately, accounting properly for the contribution to the integral due to the jumps (clearly this can get quite tedious if there are many discontinuities because the integrand changes each time the number t at which the value of the integral is desired changes). We will now illustrate how to do this in case there are discontinuities at zero. If $F(0) > 0$ or $G(0) > 0$, write

$$\begin{aligned} F * G(t) = {} & F(0)G(t) + G(0)(F(t) - F(0)) + (1 - F(0)) \\ & \cdot(1 - G(0)) \int_0^t F^0(t-u)\, dG^0(u), \end{aligned} \tag{24}$$

where $F^0(t) = (F(t) - F(0))/(1 - F(0))$, and similarly for G^0. F^0 and G^0 are now continuous at zero, so we can apply the quadrature (25) or (26a)-(26b), below, to the integral in (24), add the correction terms, and obtain the correct result.

The result of a convolution is another cdf, and generally the values of this function at more than a single point are desired (this is certainly true if repeated convolutions are being computed). Accordingly, we will assume that it is desired to have $N+1$ values of $F * G$ over some, possibly large, interval $[0, t]$. We will choose a subdivision like (7), with $N+1$ points, of this interval, and use the resulting points $\{t_i\}_{i=0}^N$ to construct compound trapezoidal and Simpson rules for (23). While there is no requirement that these subdivision points be equally spaced, it is convenient to take them so because of the difference that appears in the integrand (the equal spacing limits the number of function evaluations of F and G to $N+1$ each; unequal spacing increases the number of function evaluations needed). Let $h := t/N$; then $t_i = ih$, $i = 0(1)N$. The trapezoidal rule approximation for $F * G(t_i)$ is

$$Q^S_1(F, G)(t_i) := \sum_{k=1}^{i} \frac{F((i-k)h) + F((i-k+1)h)}{2}[G(kh) - G((k-1)h)], \tag{25}$$

for $i = 1(1)N$. The Simpson's rule approximation for $F * G(t_i)$ requires separating even and odd cases. For N even and $i = 2(2)N$, or for N odd and $i = 2(2)N-1$, we have

$$\begin{aligned} Q^S_2(F, G)(t_i) := \sum_{k=1}^{i/2} \Bigg[& \frac{F((i-2k)h) + 4F((i-2k+1)h) + F((i-2k+2)h)}{6} \\ & \cdot [G(2kh) - G((2k-2)h)] + \frac{F((i-2k)h) - F((i-2k+2)h)}{3} \end{aligned} \tag{26a}$$

$$\cdot[G(2kh)-2G((2k-1)h)+G((2k-2)h)]\Bigg].$$

For N even and $i=1(2)N-1$, or N odd and $i=1(2)N$, we have

$$\begin{aligned}Q_2^S(F,G)(t_i) := {} & \frac{F(ih)+4F((i-\frac{1}{2})h)+F((i-1)h)}{6}[G(h)-G(0)] \\ & +\frac{F((i-1)h)-F(ih)}{3}[G(0)-2G(h/2)+G(h)] \\ & +\sum_{k=1}^{(i-1)/2}\Bigg[\frac{F((i-2k-1)h)+4F((i-2k)h)+F((i-2k+1)h)}{6} \\ & \qquad \cdot[G((2k+1)h)-G((2k-1)h)] \\ & +\frac{F((i-2k-1)h)-F((i-2k+1)h)}{3} \\ & \qquad \cdot[G((2k+1)h)-2G(2kh)+G((2k-1)h)]\Bigg].\end{aligned} \tag{26b}$$

The contribution from the integral from zero to t_1 (first two terms on the right hand side of (26b)) could have been obtained from the trapezoid rule (25). However, for accuracy commensurate with that of Simpson's rule used over the rest of the interval, particularly for repeated convolution, it is recommended that Simpson's rule be used as indicated in (26b). This increases the required number of function calls for F and G, but the added accuracy is worth the small effort.

In all cases, $F*G(0)=F(0)G(0)=0$.

4.2. Error analysis with the trapezoidal and Simpson rules. In this section, we use explicitly the derivatives of $P_n x$ that appear in (14) and (15) to construct error bounds for (25) and (26a)-(26b). In a notation similar to that previously introduced, we let $B_r^i(F)$ denote $\sup\{|D^rF(u)|: 0\leqq u\leqq t_i\}$.

THEOREM 6. *If F and G have two continuous derivatives on* $[0,t]$, *then*

$$|F*G(t_i)-Q_1^S(F,G)(t_i)|\leqq\frac{t_ih^2}{12}[B_1^i(F)B_2^i(G)+B_2^i(F)B_1^i(G)]. \tag{27}$$

Proof. Begin with

$$\begin{aligned}&|F*G(t_i)-Q_1^S(F,G)(t_i)| \\ &\leqq\sum_{k=1}^{i}\Bigg|\int_{(k-1)h}^{kh}F(ih-u)\,dG(u) \\ &\quad -\frac{F((i-k)h)+F((i-k+1)h)}{2}[G(kh)-G((k-1)h)]\Bigg|.\end{aligned} \tag{28}$$

Apply (15) with $n=1$ to each term on the right hand side. $K_1^1=0$ and $K_1^2=-h^3/6$, independent of k, and from (9) we obtain

$$D_uP_1F(ih-u)=\frac{F((i-k)h)-F((i-k+1)h)}{h}.$$

This can be written as $-DF(\xi_k)$ for some ξ_k strictly between $(i-k)h$ and $(i-k+1)h$.

Inserting this into (28) yields

$$|F * G(t_i) - Q_1^S(F, G)(t_i)| \leqq \sum_{k=1}^{i} \frac{h^3}{12} |D^2F(ih - \tau_k)DG(\tau_k) + DF(\xi_k)D^2G(\eta_k)|,$$

where τ_k, ξ_k, and η_k are all strictly between $(i-k)h$ and $(i-k+1)h$. Equation (27) follows immediately, and this completes the proof of Theorem 6. $\square$

THEOREM 7. *If F and G have four continuous derivatives on* $[0, t]$, *then*

$$(29)\quad |F * G(t_i) - Q_2^S(F, G)(t_i)| \leqq \frac{t_i h^4}{180} \sum_{k=1}^{4} c_k B_k^i(F) B_{5-k}^i(G) + \frac{t_i h^5}{120} B_2^i(F) B_4^i(G),$$

where $c_1 = c_4 = 1$ *and* $c_2 = c_3 = 4$.

Proof. Begin with the analogue of (28) that uses Q_2^S and (26a)-(26b) instead of (25). Apply (14) with $n = 2$ to each term individually, noting that $K_2 = -4h^5/15$ independent of k. Explicitly evaluating $D_u P_2 F(ih - u)$ and $D_u^2 P_2 F(ih - u)$ from (9), and using the mean value theorem for divided differences, we obtain $D_u P_2 F(ih - u) = -DF(\eta_k) + D^2F(\xi_k)(u - (2k - \frac{3}{2})h)$ and $D_u^2 P_2 F(ih - u) = D^2F(\xi_k)$ for some ξ_k strictly between $(i-2k)h$ and $(i-2k+2)h$, and η_k strictly between $(i-2k+1)h$ and $(i-2k+2)h$. Inserting all this into (14), we obtain (29) upon noting that $-h/2 \leqq u - (2k - \frac{3}{2})h \leqq 3h/2$ for $(2k-2)h \leqq u \leqq 2kh$.

4.3. Choice of step size. Equations (16), (17), (27), and (29) are useful for determining the spacing h necessary to obtain a prescribed overall error. To do this, it is necessary first to evaluate the various bounds B_i, then set the appropriate expression equal to the desired overall error, and solve for h. Generally, this will turn out to be a fairly conservative procedure, and the h obtained will most likely be smaller than is really necessary to obtain the desired error bound. This is illustrated in Table 2, below,

TABLE 2
Errors for the three examples.

Time t	Example 1 Trapezoidal rule	Example 1 Simpson's rule	Example 2 Repeated Simpson	Example 3 Simpson's rule
0	0.000000000000	0.000000000000	0.000000000000	0.0000000
500	−0.000053837217	0.000000011217	−0.000000011205	0.0002117
1000	−0.000074582390	0.000000015539	−0.000000034788	0.0002645
1500	−0.000077890429	0.000000016228	−0.000000046939	0.0000865
2000	−0.000072673834	0.000000015141	−0.000000046406	0.0001369
2500	−0.000063884514	0.000000013310	−0.000000037976	0.0000486
3000	−0.000054172538	0.000000011287	−0.000000026472	0.0000823
3500	−0.000044870012	0.000000009348	−0.000000015141	0.0000301
4000	−0.000036570544	0.000000007619	−0.000000005665	0.0000526
4500	−0.000029467231	0.000000006139	0.000000001388	0.0000195
5000	−0.000023547180	0.000000004906	0.000000006092	0.0000347
5500	−0.000018701315	0.000000003896	0.000000008831	0.0000130
6000	−0.000014784619	0.000000003080	0.000000010075	0.0000234
6500	−0.000011647722	0.000000002427	0.000000010274	0.0000088
7000	−0.000009152196	0.000000001907	0.000000009801	0.0000160
7500	−0.000007176830	0.000000001495	0.000000008942	0.0000060
8000	−0.000005619098	0.000000001171	0.000000007901	0.0000111
8500	−0.000004394218	0.000000000916	0.000000006816	0.0000042
9000	−0.000003433175	0.000000000715	0.000000005771	0.0000078
9500	−0.000002680403	0.000000000558	0.000000004815	0.0000030
10000	−0.000002091530	0.000000000436	0.000000003970	0.0000055

where the true error in the computation turns out to be smaller than the error predicted by using (27) or (29) by at least two orders of magnitude.

4.4. Repeated convolution. For repeated convolutions, define $F_1 = F$, and, for $n > 1$, $F_{n+1} = F_n * F$. To evaluate F_n, use (25) or (26a)-(26b) iteratively. That is, for the trapezoidal rule approximation to $F_n(t_i)$, use $Q_1^S(F_{n-1}, F)(t_i)$, and for the Simpson's rule approximation, use $Q_2^S(F_{n-1}, F)(t_i)$. Since (25) and (26a)-(26b) compute the values of F_{n-1} at the same mesh points at which the original function F is evaluated, the equally spaced mesh is particularly convenient for this iterated operation.

Using the Simpson's rule requires $F_{n-1}(h/2)$, which in turn requires $F_{n-2}(h/4)$, and so on down to $F(h/2^{n-1})$. These, as well as the values $F(h/2^{n-1}), \cdots, F(h/2)$, are used to obtain $F_n(h) = F_n(t_1)$. Once $F_n(t_1)$ is obtained, these values are no longer needed for the rest of the computation of $F_n(t_i)$, $i \geqq 2$.

To compute $F_j * G_k$, first compute F_j and G_k individually, and then use (25) or (26a)-(26b) to combine them. Again, when using the Simpson's rule, the values $F_{j-i}(h/2^i)$, $i = 1(1)j-1$, and $G_{k-i}(h/2^i)$, $i = 1(1)k-1$, are needed.

4.5. Examples. In this section, we give three examples that illustrate numerical convolution of life distributions using the trapezoidal and Simpson rules for Stieltjes integrals. The first example compares the trapezoidal with the Simpson rule on a single convolution, the second example uses the Simpson rule to evaluate a repeated convolution, and the third example illustrates the convolution, using (26a)-(26b) directly, of two life distributions having densities that are singular at the origin. We have chosen illustrations using exponential and gamma distributions so we could easily make comparisons between the computed values of the convolution and the true values that are obtainable in these cases in closed form.

The first example is the convolution of $F(t) = 1 - e^{-t/1000}$ with $G(t) = 1 - e^{-t/2000}$. In this case, $F * G(t) = 2G(t) - F(t)$. Table 2 displays the error in the trapezoidal rule approximation (25) for $F * G$ (column 2), and the error in the Simpson's rule approximation (26a)-(26b) (column 2). We have omitted the true and computed values in all the examples from Table 2 to save typesetting pain. The author will be happy to send tables of those values upon request. In evaluating this convolution, we used $a = 0$, $b = 10{,}000$, $h = 50$, and $N = 200$. To save space, the table shows only every tenth point. The computations were done in double precision, using the C compiler on a VAX 8600 running the UNIX System V Release 2 operating system. The largest error in Example 1 occurs at $t = 1500$ for both the trapezoidal and Simpson rules. The maximum of the absolute value of the error predicted by (27) at $t = 1500$ for the trapezoidal rule evaluation is 2.3438E-4, and for the Simpson rule, it is 1.0767E-7 from (29). The worst case in the table is $t = 1500$, because the error bounds (27) and (29) increase linearly with t, while for $t > 1500$, the errors as seen in the table decrease.

The second example concerns a repeated convolution. Using the F and G above, we compute $F_3 * G$ using the Simpson rule and the iterative procedure suggested in § 4.3. Again, we use $a = 0$, $b = 10{,}000$, $h = 50$, and $N = 200$, and double precision arithmetic. The errors, again only for every tenth point, are shown in column 4 of Table 2. While no explicit error analysis like (27) or (29) is given in this paper for repeated convolution using the Simpson's rule (26a)-(26b), examination of Table 2 shows that respectably small errors are achievable with this procedure.

In the third example, the two distributions to be convolved each have densities with singularities at the origin: the tangent (from the right) to the distribution is vertical in each case. In this example, F is a gamma distribution with location parameter 3×10^{-4} (units of hr^{-1}) and shape parameter $-\frac{3}{4}$, and G is a gamma distribution with the same location parameter and shape parameter $-\frac{1}{2}$ (see [16] for the parametrization of the gamma distribution used here). The mean of F is 833.33 hours, and the mean of G is 1666.67 hours. Then $F * G$ is also a gamma distribution with the same location parameter, and with shape parameter $-\frac{1}{4}$ ([16], p. 209); its mean is 2500 hours. The last column of Table 2 shows the error obtained from numerical evaluation of this convolution using (26a)-(26b). Again, we use $a=0$, $b=10{,}000$, $h=50$, and $N=200$, and double precision arithmetic, and display only every tenth point. While (29) does not apply here because of the singularities, the last column of Table 2 shows that (26a)-(26b) still produces reasonable results, especially when considering the simplicity of the procedure as compared to the methods recommended in [4] and [6].

5. Conclusions. This paper provides new methods for the evaluation of Stieltjes integrals by developing analogues of the closed Newton-Cotes quadrature rules. These are based on a simple idea and are most useful when it is impossible or undesirable to use derivatives of the integrator function in the quadrature. Error analyses are given that aim to show the highest order achievable by the rules. An error analysis under weaker conditions than those in Theorems 2 and 3 is presented for the trapezoidal rule. It would be useful to obtain error analyses under these weaker conditions for the higher-order rules as well. A major virtue of these rules is their simplicity, and, especially for the lower-degree rules, robustness. The trapezoidal rule can be adapted to a Romberg scheme provided it is possible to find a sequence of argument values for which the differences of the integrator function are equal from one argument value to the next.

The trapezoidal and Simpson rules are used to develop two algorithms for numerical convolution of life distributions. These algorithms are usable even in cases where previous algorithms [2], [4] fail, such as for convolutions of Weibull distributions with shape parameters less than one-half. Explicit error analyses are provided for single convolution. Four numerical convolution examples are computed, providing comparisons between computed values and known values in these cases.

Acknowledgments. I thank my colleagues C. L. Gibson and D. L. Jagerman for their contributions to this paper, including discussions of the ideas and assistance with computer implementation. I would also like to thank the referee and the editor for several suggestions that have contributed to the final version's improved quality over the original submission.

REFERENCES

[1] M. F. NEUTS, *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach*, The Johns Hopkins University Press, Baltimore, MD, 1981.

[2] R. CLEROUX AND D. J. MCCONALOGUE, *A numerical algorithm for recursively-defined convolution integrals involving distribution functions*, Management Sci., 22 (1976), pp. 1138-1146.

[3] D. J. MCCONALOGUE, *Convolution integrals involving probability distribution functions (algorithm* 102), Comput. J., 21 (1978), pp. 270-272.

[4] ———, *Numerical treatment of convolution integrals involving distributions with densities having singularities at the origin*, Commun. Statist.-Simula. Computa., B10 (1981), pp. 265-280.

[5] M. ABRAMOWITZ AND I. A. STEGUN, *Handbook of mathematical functions*, Vol. 55, in the National Bureau of Standards Applied Mathematics Series, U.S. Government Printing Office, Washington, DC, 1964.

[6] L. A. BAXTER, *Some remarks on numerical convolution*, Commun. Statist.-Simula. Computa., B10 (1981), pp. 281-288.

[7] J. A. SHOHAT AND J. D. TAMARKIN, *The Problem of Moments*, American Mathematical Society, Providence, RI, 1943.

[8] Z. SCHULTEN, D. G. M. ANDERSON, AND R. G. GORDON, *An algorithm for the evaluation of the complex Airy functions*, J. Comp. Phys., 31 (1979), pp. 60-75.

[9] W. GAUTSCHI, *On generating orthogonal polynomials*, SIAM J. Sci. Statist. Comput., 3 (1982), pp. 289-317.

[10] V. I. KRYLOV, *Priblizhennoe vychislenie integralov* (*Approximate Calculation of Integrals*), A. H. Stroud, transl., Macmillan, New York, 1962.

[11] M. TORTORELLA, *Numerical solution of integral equations of renewal type*, AT&T Bell Laboratories Quality Assurance Center Research Report no. 90-1, Whippany, NJ, 1990, to appear.

[12] E. ISAACSON AND H. B. KELLER, *Analysis of Numerical Methods*, John Wiley, New York, 1966.

[13] G. G. LORENTZ, *Approximation of Functions*, Holt, Rinehart, and Winston, New York, 1966.

[14] M. J. D. POWELL, *Approximation Theory and Methods*, Cambridge University Press, Cambridge, MA, 1981.

[15] A. H. STROUD, *Numerical quadrature and solution of ordinary differential equations*, in Applied Mathematical Sciences, Vol. 10, Springer-Verlag, Berlin, NY, 1974.

[16] M. F. NEUTS, *Probability*, Allyn and Bacon, Boston, 1973.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 4, pp. 749-766, July 1990

SMOOTHING POLYNOMIAL SPLINES FOR BIVARIATE DATA*

STEWART J. ANDERSON†, RICHARD H. JONES‡, AND GEORGE D. SWANSON§

Abstract. An extension of the smoothing polynomial spline to fit bivariate response data is presented. The data are modeled as integrated random walks with observational errors. Correlation can exist in the random walks, the observational errors, or both. The Kalman filter is used to calculate the log likelihood of the data as a function of the unknown parameters in the covariance matrices, and nonlinear optimization is used to obtain maximum likelihood estimates of the parameters. A modification of the Kalman filter is used at the beginning of the data to allow the use of diffuse (noninformative) priors. This model is applied to the problem of characterizing gas exchange time series of exercising subjects.

Key words. smoothing polynomial splines, vector splines, bivariate response models, Kalman filter, maximum likelihood estimation, integrated random walks, gas exchange measurements, Fieller's theorem

AMS(MOS) subject classification. 62-07

1. Introduction.

The use of piecewise continuous functions called *splines* to model data is an old concept (Thiele (1880)). This technique has provided an alternative to simple linear or nonlinear regression when data cannot be modeled using a single functional form. Until the 1970's, much of the work in splines assumed that the data span could be divided into subintervals so that in each subinterval the data could be modeled by a prespecified function (Wold (1974)). It was also usually assumed that the specified functions were connected to each other so that overall continuity was maintained. The points where successive functions connected were called *knots*. The number of knots in most of the work prior to 1970 was assumed to be very small in comparison to the number of data points. Later, a type of spline that has knots at every data point was introduced. That is, if n data pairs, (t_i, y_i), are being modeled such that the t_i are distinct, then the assumption is that $n-1$ functions will model the data. In some cases, the functions will not all be unique. For our purposes, the functions are assumed to be polynomials of degree $2m-1$, where m is a positive integer. Successive functions are pieced together with the assumption that the first $m-1$ derivatives of the functions are continuous at the knot. These splines are known as *smoothing polynomial splines.*

In this paper, we extend the smoothing polynomial spline model to the case where a vector, $(y_{1i}, y_{2i})^T$, $i=1, \cdots, n$ of observations is modeled over time t_i. We assume that the errors associated with the vector model have a Gaussian structure with correlation. We also assume that there is correlation in the underlying stochastic structure of the model. Thus, we used a stochastic motivation of the model and use a likelihood criterion to arrive at our "best model."

In § 2 of this paper, we give a brief review of univariate smoothing polynomial splines. A brief overview of the Kalman recursion and Rauch-Tung-Striebel fixed interval smoothing is given in § 3. We then use the univariate splines to model gas

* Received by the editors December 30, 1987; accepted for publication (in revised form) June 20, 1989. This research was partially supported by the National Science Foundation, Division of Mathematical Sciences, grant DMS-8702195, and the National Institute of General Medical Studies grant R01-GM38519.

† Department of Biostatistics, Graduate School of Public Health, University of Pittsburgh, Pittsburgh, Pennsylvania 15261.

‡ Department of Preventive Medicine and Biometrics, University of Colorado Health Sciences Center, Denver, Colorado 80262.

§ Department of Anesthesiology, University of Colorado Health Sciences Center, Denver, Colorado 80262, and Department of Physical Education, Chico State University, Chico, California 95929.

exchange data on exercising subjects in § 4 of the paper. In § 5, we then introduce a stochastically motivated two-dimensional vector model. In § 6, we examine some of the assumptions of the vector model. In § 7, we show how the vector model was used on the same data introduced in § 4. Finally, in § 8, we give some brief conclusions and directions for further research.

2. Univariate smoothing polynomial splines. A notational convention to be used in this section is as follows: the notation $x^{(m)}(t)$ denotes the mth derivative of $x(t)$, whereas $X^{(m)}(t)$ denotes the mth integral of $X(t)$.

Much of the theory behind the smoothing polynomial spline was developed by Wahba (Kimeldorf and Wahba (1970) and Wahba (1978)). To choose the smoothing parameter (equivalent to choosing the signal-to-noise ratio), Craven and Wahba (1979) utilized a method called *Generalized Cross-Validation* (GCV) to find the optimal polynomial spline given a particular set of data. Although the GCV method was robust (that is, could estimate the smoothing parameter whether or not the unknown function came from a stochastic process), the associated numerical algorithm tended to be somewhat slow. Weinert, Byrd, and Sidhu (1980) and later, Wecker and Ansley (1983) showed that by assuming an underlying process of an integrated random walk with observational error, a faster algorithm could be developed. This algorithm represented the model in a state space form. It utilized the *Kalman filter* (Kalman (1960)) for calculating the likelihood function (Schweppe (1965)), and nonlinear optimization was used to obtain maximum likelihood estimates.

More recently, $O(n)$ algorithms have been developed for computing the smoothing spline with GCV (Ansley and Kohn (1987) and Hutchinson and de Hoog (1985)). However, for our univariate and two-dimensional vector smoothing polynomial splines, we use the stochastically motivated approach to estimate optimum smoothing parameters.

In regression analysis, the relationship between n data points, (t_i, y_i), is described by a fixed functional form, $y_i = f(t_i) + e_i$, $i = 1, \cdots, n$, where the e_i are usually independently and identically distributed $N(0, R)$. This functional form is usually specified prior to fitting the data. In some cases, however, no particular functional form can readily be found to adequately describe a relationship between two observed variables. In this case, it is appropriate to use *smoothing polynomial splines* to characterize a relationship. One advantage to this approach is that no prespecified functional form must be used. Smoothing polynomial splines have an added advantage in that they are equivalent to many fixed functional forms. Thus, they can be used whether or not an underlying function is known for a particular process.

Schoenberg (1964) showed that the smoothing spline, $g_{n,\kappa}(t_i)$, of degree $2m-1$ minimizes the quantity

$$(1)\qquad r = n^{-1}\sum_{i=1}^{n}[g(t_i)-y_i]^2 + \kappa\int_0^1 (g^{(m)}(u))^2\,du.$$

Note that if $\kappa \to 0$, this criterion gives an interpolating spline whereas if $\kappa \to \infty$, then the criterion gives a polynomial of degree $m-1$. Therefore, the criterion is a measure of the deviation of $g_{n,\kappa}$ from the span of polynomials of degree less than m (Wahba (1978)). Wahba (1978) showed that the stochastic process

$$(2)\qquad g_{n,\kappa}(t) = \sum_{k=0}^{m-1}\alpha_k\frac{(t-a)^k}{k!} + R^{1/2}\sigma\int_a^t \frac{(t-h)^{m-1}}{(m-1)!}\,dW(h),$$

where $W(h)$ is a Wiener process with unit variance, is the solution to minimizing $r(t)$ in equation (1) with $\kappa = 1/(n\sigma)$. This is conditional on letting $\alpha = (\alpha_0, \alpha_1, \cdots, \alpha_{m-1})^T$

have a diffuse prior, that is, $\boldsymbol{\alpha}(0) \sim N(\boldsymbol{\mu}, \xi \mathbf{I}_m)$ where $\boldsymbol{\mu}$ is arbitrary, $\mathbf{I}_m$ is the $m \times m$ identity matrix and $\xi \to \infty$. Wecker and Ansley then introduced the stochastic process,

$$(3) \qquad X^{(j)}(t) = \sum_{k=0}^{j-1} X^{(m-k)}(a) \frac{(t-a)^k}{k!} + R^{1/2}\sigma \int_a^t \frac{(t-h)j^{i-1}}{(j-1)!} dW(h),$$

$j = m, \cdots, 1$. If $\mathbf{X}(t) = (X^{(m)}(t), \cdots, X^{(1)}(t))^T$, then Wahba's model (2) can be rewritten as

$$(4) \qquad g_{n,\kappa}(t_i) = X^{(m)}(t),$$

where $\alpha_k = X^{(m-k)}(t)$ and $\mathbf{X}(a)$ is the starting condition of the process $\mathbf{X}(t)$. It is usually convenient to set $a = t_1$. Note that the "state," $\mathbf{X}(t)$, consists of $X^{(m)}(t)$ and its first $m-1$ derivatives. The state vector can also be written as $\mathbf{x}(t) = (x(t), \cdots, x^{(m)}(t))^T$, where $x(t) = X^{(m)}(t)$.

The significance of equations (1)-(3) is the assumption that the process underlying our polynomial spline model is driven by noise in the $(m-1)$st derivative. For example, in a cubic spline, the noise in the 1st derivative is translated to noise in the position component. This complete process allows flexibility because the parameters in the state vector can change over time. Therefore, a local effect can be modeled as well as a global effect.

The estimation of the state vectors uses all of the available data. This requires that the estimation of each state vector, $\mathbf{x}(t)$, be carried out in two steps. The first step, called *filtering*, uses all of the data up to and including the data point at t_i to estimate the state vector (it is assumed that the t_i's are ordered so that $t_i < t_{i+1}$, $i = 1, \cdots, n-1$). The second step, called *smoothing*, incorporates the rest of the available data into another estimate that is then weighted with the filtered estimate to give a final estimate that is based on all of the available data.

For either a continuous or discrete model, an assumption of the Kalman filter is that a given model can be represented in a state space form. Given n data pairs, (t_1, y_1), $(t_2, y_2), \cdots, (t_n, y_n)$, a state space model can be written as:

$$(5) \qquad \mathbf{x}(t_{i+1}) = \boldsymbol{\Phi}(\delta_i)\mathbf{x}(t_i) + \mathbf{U}(t_i),$$

and,

$$(6) \qquad \mathbf{y}(t_i) = \mathbf{H}(t_i)\mathbf{x}(t_i) + \mathbf{v}(t_i),$$

where $\delta_i = t_i - t_{i-1}$, $\mathbf{x}(t_i)$ is the state vector at time t_i, $\boldsymbol{\Phi}(\delta_i)$ is the state transition matrix, $\mathbf{y}(t_i)$ is a vector of observations, $\mathbf{H}(t_i)$ is a matrix of constants, and $\mathbf{U}(t_i)$ and $\mathbf{v}(t_i)$ are random input matrices. For the smoothing polynomial spline (or continuous integrated random walk) model (Wecker and Ansley (1983) and Jones and Tryon (1987)),

$$(7) \qquad \boldsymbol{\Phi}(\delta_i) = \exp(\mathbf{F}\delta_i) = \mathbf{I} + \sum_{i=1}^{m-1} \frac{(\mathbf{F}\delta_i)^i}{i!} = \begin{bmatrix} 1 & \delta_i & \delta_i^2/2 & \cdots & \delta_i^{m-1}/(m-1)! \\ 0 & 1 & \delta_i & \cdots & \delta_i^{m-2}/(m-2)! \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \delta_i \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}.$$

Also, given $\boldsymbol{\Phi}(\delta_i)$, we can solve for the covariance matrix of the random input from time t_{i-1} to t_i (Kohn and Ansley (1987) and Anderson (1987)). Denoting the jkth element of this covariance matrix by $Q_{jk}(\delta_i)$, we can write

$$(8) \qquad Q_{jk}(\delta_i) = \frac{\delta_i^{2m-j-k+1}\sigma^2}{(m-j)!(m-k)!(2m-j-k+1)},$$

$i=1, \cdots, n$; $j=1, \cdots, m$; and $k=1, \cdots, m$. Also, for the univariate continuous random integrated walk model, $\mathbf{x}(t_i \mid t_i)=(x(t_i \mid t_i), x'(t_i \mid t_i), \cdots, x^{(m-1)}(t_i \mid t_i))^T$, and $\mathbf{H}=(1, 0, \cdots, 0)$. In addition, the likelihood function will be maximized with respect to the variance of the stochastic process, σ^2, and the observational noise variance, R. The Kalman filter is used to calculate $-2 \ln$ likelihood for given values of σ^2 and R. This is embedded as a function evaluation in a nonlinear optimization routine to obtain maximum likelihood estimates of σ^2 and R, which in turn give estimates of the optimal amount of smoothing under our model assumptions.

3. The Kalman recursion and Rauch–Tung–Striebel fixed-interval smoothing. To begin the Kalman recursion, we will assume that the stochastic process $\mathbf{x}(t)$ has some distribution prior to gathering the data. To reflect our lack of prior knowledge, we use a diffuse prior distribution suggested by Wahba (1978). That is, we will assume $\mathbf{x}(0 \mid 0) \sim \mathbf{N}(\mathbf{0}, \lambda \mathbf{I}_\mathbf{m})$, $\lambda \to \infty$, where $\mathbf{I}_\mathbf{m}$ denotes the $m \times m$ identity matrix. Note that $\mathbf{x}(0 \mid 0)$ denotes the a priori estimate of $\mathbf{x}(t)$. Since the model is nonstationary, we must modify the first few steps of the ordinary recursion to incorporate the infinite variance of the prior distribution (Kohn and Ansley (1985), (1987) and de Jong (1988)). Assuming these modifications have been made (see Appendix A), the Kalman recursion is as follows:

(1) Calculate a one-step prediction:

$$\mathbf{x}(t_i \mid t_{i-1})=\mathbf{\Phi}(\delta_i)\mathbf{x}(t_{i-1} \mid t_{i-1})$$

(2) Calculate the covariance matrix of the one-step prediction:

$$\mathbf{P}(t_i \mid t_{i-1})=\mathbf{\Phi}(\delta_i)\mathbf{P}(t_{i-1} \mid t_{i-1})\mathbf{\Phi}^T(\delta_i)+\mathbf{Q}(\delta_i)$$

(3) Predict the next observation:

$$\mathbf{Y}(t_i \mid t_{i-1})=\mathbf{H}\mathbf{x}(t_i \mid t_{i-1})$$

(4) Calculate the *innovation* vector:

$$\mathbf{I}(t_i)=\mathbf{y}(t_i)-\mathbf{Y}(t_i \mid t_{i-1})$$

(5) Calculate the innovation covariance matrix:

$$\mathbf{V}(t_i)=\mathbf{H}\mathbf{P}(t_i \mid t_{i-1})\mathbf{H}^T+\mathbf{R}$$

(6) The contribution of the innovation to $-2 \ln$ likelihood is:

$$\mathbf{I}^T(t_i)\mathbf{V}^{-1}(t_i)\mathbf{I}(t_i)+\ln |\mathbf{V}(t_i)|$$

(7) Calculate the *Kalman gain* matrix:

$$\mathbf{K}(t_i)=\mathbf{P}(t_i \mid t_{i-1})\mathbf{H}^T\mathbf{V}^{-1}(t_i)$$

(8) Update the estimate of the state vector:

$$\mathbf{x}(t_i \mid t_i)=\mathbf{x}(t_i \mid t_{i-1})+\mathbf{K}(t_i)\mathbf{I}(t_i)$$

(9) Update the estimate of the covariance matrix:

$$\mathbf{P}(t_i \mid t_i)=\mathbf{P}(t_i \mid t_{i-1})-\mathbf{K}(t_i)\mathbf{H}\mathbf{P}(t_i \mid t_{i-1}).$$

The Kalman filter gives an optimal estimate (in the least squares sense) of $\mathbf{x}(t_i)$ given $(t_1, y_n), \cdots, (t_i, y_i)$ (Kalman (1960)). The problem of finding optimal estimates of $\mathbf{x}(t_i)$ given $\mathbf{y}(t_1), \cdots, y(t_n)$, $(n > i)$ is called a *smoothing* problem. Thus, the solution to the smoothing problem will give us the optimal least squares estimates of $\mathbf{x}(t_i)$, $i=1, \cdots, n$ given *all* of the data. In our problem, we are interested in obtaining these estimates with the assumption that our data span is fixed. Our model is continuous; however, our sampling intervals are discrete. Therefore, we will use *discrete fixed-interval* smoothing (Brown (1983)).

Rauch, Tung, and Striebel (1965) solved the discrete fixed-interval smoothing problem. Their solution consisted of two steps: (I) A forward sweep of the data using the Kalman filter; and (II) A backward sweep of the data which, when weighted with the forward sweep, gives a minimum least squares estimate for each $\mathbf{x}(t_i)$, $i=1,\cdots,n$ given all of the data, $(t_1,\mathbf{y}_1),\cdots,(t_n,\mathbf{y}_n)$. A rigorous derivation of the solution is given in Rauch, Tung, and Striebel (1965) and Sage and Melsa (1971). A summary of the solution is as follows:

(1) Obtain the Kalman filter estimates $\mathbf{x}(t_{i+1}|t_i)$, $\mathbf{x}(t_i|t_i)$, $\mathbf{P}(t_{i+1}|t_i)$, $\mathbf{P}(t_i|t_i)$ for $i=1,\cdots,n-1$;

(2) Use $\mathbf{x}(t_n|t_n)$ and $\mathbf{P}(t_n|t_n)$ as initial estimates for the backward sweep;

(3) Predict $\mathbf{x}(t_i)$, $i=n-1,\cdots,1$ as

$$\mathbf{x}(t_i|t_n)=\mathbf{x}(t_i|t_i)+\mathbf{A}(t_i)[\mathbf{x}(t_{i+1}|t_n)-\mathbf{x}(t_{i+1}|t_i)], \tag{9}$$

where

$$\mathbf{A}(t_i)=\mathbf{P}(t_i|t_i)\mathbf{\Phi}^T(\delta_{i+1})\mathbf{P}^{-1}(t_{i+1}|t_i); \tag{10}$$

(4) Predict the covariance matrix of $\mathbf{x}(t_i)$ as:

$$\mathbf{P}(t_i|t_n)=\mathbf{P}(t_i|t_i)+\mathbf{A}(t_i)[\mathbf{P}(t_{i+1}|t_n)-\mathbf{P}(t_{i+1}|t_i)]\mathbf{A}^T(t_i). \tag{11}$$

The notation convection here is that, for example, $\mathbf{x}(t_i|t_n)$ is the estimate of $\mathbf{x}(t_i)$ given $\mathbf{y}(t_1),\cdots,\mathbf{y}(t_n)$.

To conserve the amount of memory needed, one can store the smoothed estimates, $\mathbf{x}(t_i|t_n)$, $\mathbf{x}(t_{i+1}|t_n)$, $\mathbf{P}(t_i|t_n)$, and $\mathbf{P}(t_{i+1}|t_n)$ in the same arrays as their filtered counterparts, $\mathbf{x}(t_i|t_i)$, $\mathbf{x}(t_{i+1}|t_i)$, $\mathbf{P}(t_i|t_i)$, and $\mathbf{P}(t_{i+1}|t_i)$.

4. Applications of univariate smoothing splines. This application uses smoothing polynomial splines to characterize the gas exchange of a subject undergoing an exercise stress test (Wade et al. (1988)). In our first example, subject B.D. was introduced to a progressive stress test where the work load was incremented by 33 watts every three minutes. The monitoring of the subject's gas exchange measures began approximately five minutes after the beginning of the experiment. The gas exchange measures were collected at every breath. The data were then adjusted for estimates of effective lung volume and pulmonary blood flow (Sherrill (1987)). We then modeled the adjusted breath-by-breath carbon dioxide production ($\dot{V}_{CO_2}$) and oxygen consumption ($\dot{V}_{O_2}$) versus time using cubic and quintic smoothing polynomial splines. The results of the two analyses are summarized in Table 1. The $\sqrt{\hat{R}}$ and $\hat{\sigma}$ in the table refer to the estimated standard deviations of the noise terms associated with the observational error and the $(m-1)$-fold integrated Wiener process, respectively.

TABLE 1
Analysis of spline models for subject B.D.

Curve	Model	−2 ln likelihood	$\sqrt{\hat{R}}$	$\hat{\sigma}$
V_{O_2} vs. TIME	Cubic spline	4910.7	81.59	379.59
V_{O_2} vs. TIME	Quintic spline	4935.9	82.20	967.20
V_{CO_2} vs. TIME	Cubic spline	4886.7	75.64	674.10
V_{CO_2} vs. TIME	Quintic spline	4931.3	77.48	2325.3

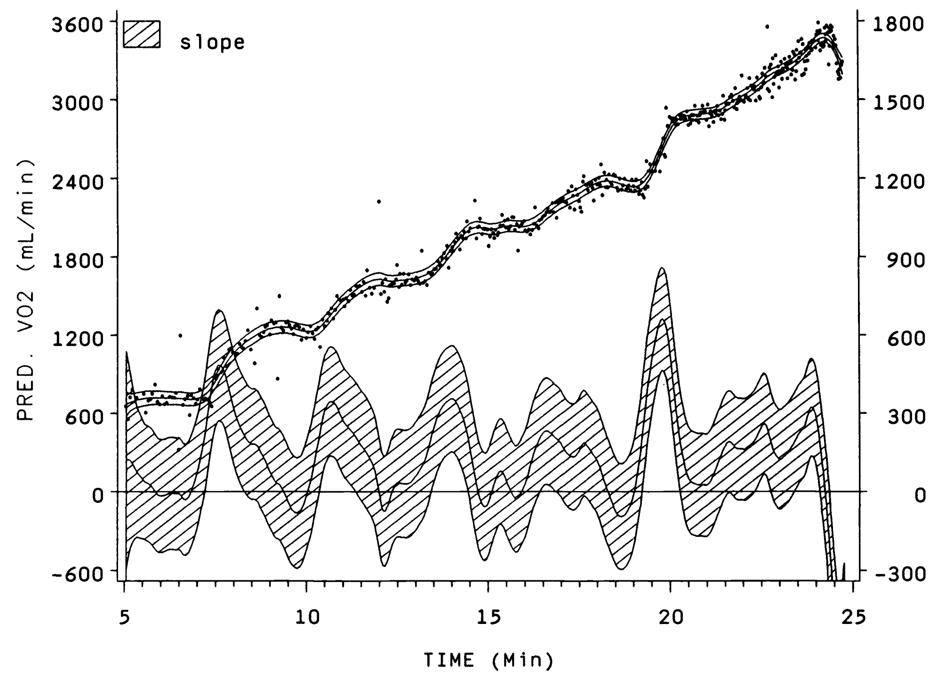


FIG. 1. *Carbon dioxide production as a function of time under increasing work for subject* B.D. *as modeled by a cubic smoothing spline. The lower curve with the wider confidence interval shows the estimated slope.*

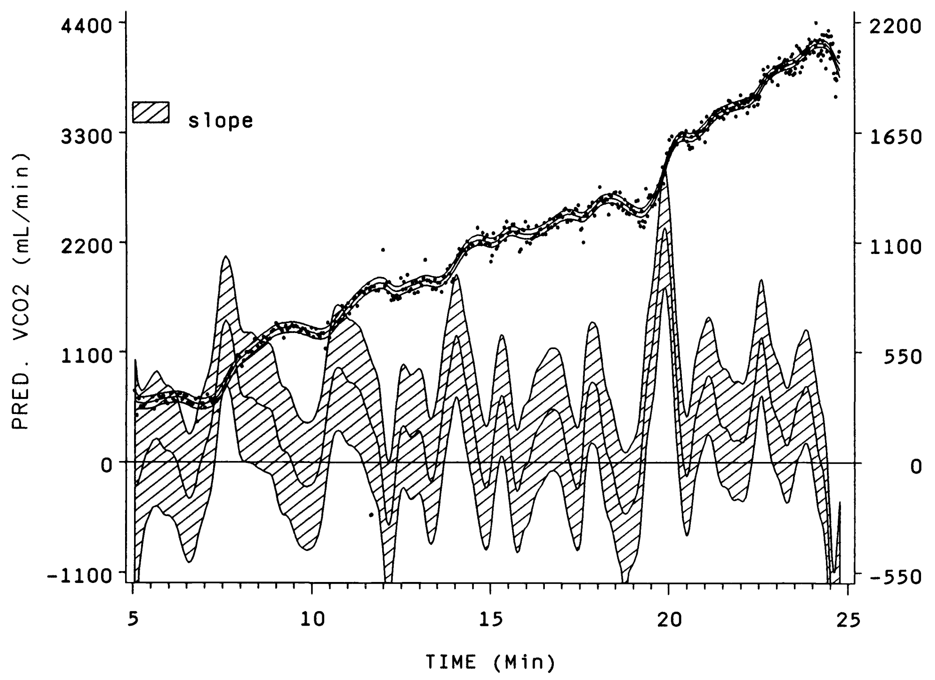


FIG. 2. *Oxygen consumption as a function of time under increasing work for subject* B.D. *as modeled by a cubic smoothing spline. The lower curve with the wider confidence interval shows the estimated slope.*

In both analyses, the cubic splines had lower values of $-2 \ln$ likelihood. Thus, since both models have the same number of parameters, cubic splines were used. The results of the two analyses are shown graphically in Figs. 1 and 2. In both figures one can see that the gas exchange curves fluctuated considerably. The effects of incrementing the work rate every three minutes are obvious on the slope curve.

As we see in Figs. 1 and 2, both the $\dot{V}_{O_2}$ and $\dot{V}_{CO_2}$ are associated with a large amount of variability with respect to time. Thus, we would need to account for the variability in both measures. Also, since the two measures rise and fall in roughly the same patterns, we would need to account for the correlation of the two processes.

Figure 3 shows similar curves for subject D.S. where the workload was increased steadily instead of in a stepwise fashion. As for subject B.D., we found that the "best" smoothing spline (using the maximum likelihood criterion) was a cubic spline.

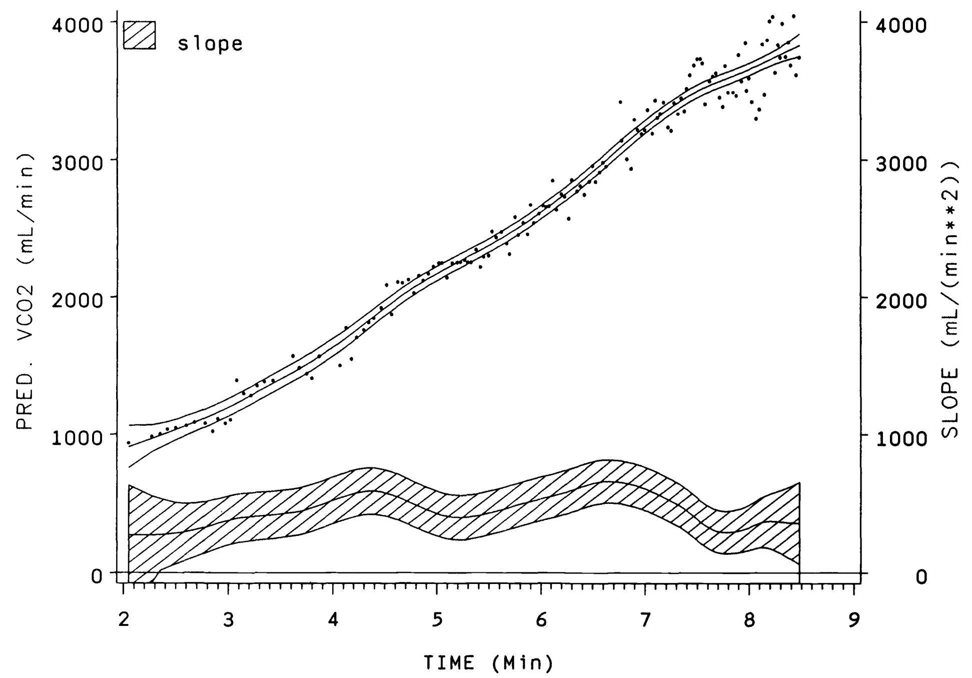


FIG. 3. *Carbon dioxide production as a function of time under increasing work for subject* D.S. *as modeled by a cubic smoothing spline. The lower curve with the wider confidence interval shows the estimated slope.*

Information about a subject's data can be lost by modeling it with a simple parametric model when the true underlying process is not well known. This loss of information can be great in situations where a large number of data points are measured. We feel that a smoothing polynomial spline model is especially useful in this setting because it can provide much information about a subject during stress testing. Beaver, Wasserman, and Whipp (1986) recommend that breadth-by-breath gas exchange data are best characterized by a plot of $\dot{V}_{CO_2}$ versus $\dot{V}_{O_2}$ which requires the use of a vector spline.

5. Introduction to two-dimensional vector smoothing polynomial splines. It is often of interest to an investigator to examine two or more variables as a function of a third variable, for example, time. The variables of interest each may be modeled univariately over time. However, in many cases, there may be underlying reasons to believe that the two or more variables may be correlated. Thus, if an investigator is interested in describing the overall trend of the two response variables over time, he or she would want to use a bivariate model. Here, "bivariate" refers to the response variable that consists of a two-dimensional vector at each time point. As in the univariate case, it may be difficult to model some data sets using a fixed parametric form. Thus, we introduce a two-dimensional vector smoothing polynomial spline.

In our formulation of a vector smoothing spline, we assume that two processes are each assocated with an underlying uncertainty (sometimes called "*plant* noise" by engineers) and an uncertainty associated with observation. These noise terms are assumed to be distributed normally. Furthermore, we assume that the underlying noise terms of the two processes are correlated and/or the observational noise terms are correlated. The state equation of the model can be written as follows:

$$\frac{d\mathbf{x}_\mathbf{B}(t)}{dt} = (\mathbf{I}_2 \otimes \mathbf{F})\mathbf{x}_\mathbf{B}(t) + \mathbf{G}_\mathbf{B}\frac{d\mathbf{W}_\mathbf{B}(t)}{dt} \tag{12}$$

where $\mathbf{I}_2$ is the 2×2 identity matrix, $\otimes$ is the Kronecker product,

$$\mathbf{x}_\mathbf{B}(t) = \begin{bmatrix} x_1(t) \\ \vdots \\ x_1^{(m-1)}(t) \\ x_2(t) \\ \vdots \\ x_2^{(m-1)}(t) \end{bmatrix}, \qquad \mathbf{G}_\mathbf{B} = \begin{bmatrix} 0 & 0 \\ \vdots & \vdots \\ G_{11} & 0 \\ 0 & 0 \\ \vdots & \vdots \\ G_{12} & G_{22} \end{bmatrix},$$

$$\mathbf{F} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix},$$

and

$$d\mathbf{W}_\mathbf{B}(t) = \begin{bmatrix} dW_1(t) \\ dW_2(t) \end{bmatrix} \sim \mathbf{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right). \tag{13}$$

From equations (12) and (13), we can conclude

$$\mathbf{G}_\mathbf{B} d\mathbf{W}_\mathbf{B}(t) \sim \mathbf{N}\left(\begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \cdots & \vdots & \vdots & \cdots & \vdots \\ 0 & \cdots & G_{11}^2 & 0 & \cdots & G_{11}G_{12} \\ 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \cdots & \vdots & \vdots & \cdots & \vdots \\ 0 & \cdots & G_{11}G_{12} & 0 & \cdots & G_{12}^2 + G_{22}^2 \end{bmatrix}\right). \tag{14}$$

The observation equation is then written as

$$\mathbf{y}_\mathbf{B}(t) = \mathbf{H}\mathbf{x}_\mathbf{B}(t) + \mathbf{v}_\mathbf{B}(t), \tag{15}$$

where

$$\mathbf{y}_\mathbf{B}(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix}, \qquad \mathbf{H} = \begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 1 & \cdots & 0 \end{bmatrix},$$

and

$$\mathbf{v_B}(t)=\begin{bmatrix} v_1(t) \\ v_2(t)\end{bmatrix}\sim \mathbf{N}\left(\begin{bmatrix}0\\0\end{bmatrix},\begin{bmatrix} U_{11}^2 & U_{11}U_{12} \\ U_{11}U_{12} & U_{12}^2+U_{22}^2\end{bmatrix}\right).$$

The factored form of the above matrices assures that the estimates of the matrices are nonnegative definite.

The extension of the univariate model to its two-dimensional vector analog requires that instead of estimating two parameters, we must now estimate up to six parameters. That is, given the data triplets, $(t_1, y_{11}, y_{21}), \cdots, (t_n, y_{1n}, y_{2n})$, we are interested in obtaining maximum likelihood estimates for $\sigma_{11}=G_{11}^2$, $\sigma_{12}=\sigma_{21}=G_{11}G_{12}$, $\sigma_{22}=G_{12}^2+G_{22}^2$, $R_{11}=U_{11}^2$, $R_{12}=R_{21}=U_{11}U_{12}$, and $R_{22}=U_{12}^2+U_{22}^2$. These estimates were obtained using a natural extension of the recursive approach as described in §§ 2 and 3. For example, in the two-dimensional vector model, the state transition matrix, $\mathbf{\Phi}(\delta_i)$ is written as

$$\mathbf{\Phi}(\delta_i)=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\otimes\begin{bmatrix} 1 & \delta_i & \delta_i^2/2 & \cdots & \delta_i^{m-1}/(m-1)! \\ 0 & 1 & \delta_i & \cdots & \delta_i^{m-2}/(m-2)! \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \delta_i \\ 0 & 0 & 0 & \cdots & 1\end{bmatrix}, \tag{16}$$

where $\delta_i=t_i-t_{i-1}$. The covariance matrix of the random input is obtained by solving

$$\mathbf{Q}(\delta)=\int_t^{t+\delta}\mathbf{\Phi}(t+\delta-\tau)\mathbf{G}\mathbf{G}^T\mathbf{\Phi}^T(t+\delta-\tau)\,d\tau. \tag{17}$$

Given the time points, $t_1, \cdots, t_n$, the solution to this integral equation is

(18)

$$\mathbf{Q}(\delta_i)=\begin{bmatrix}\sigma_{11} & \sigma_{12}\\ \sigma_{12} & \sigma_{22}\end{bmatrix}\otimes\begin{bmatrix} \dfrac{\delta_i^{2m-1}}{((m-1)!)^2(2m-1)} & \dfrac{\delta_i^{2m-2}}{(m-1)!(m-2)!(2m-2)} & \cdots & \dfrac{\delta_i^m}{m!} \\ \dfrac{\delta_i^{2m-2}}{(m-2)!(m-1)!(2m-2)} & \dfrac{\delta_i^{2m-3}}{((m-2)!)^2(2m-3)} & \cdots & \dfrac{\delta_i^{m-1}}{(m-1)!} \\ \vdots & \vdots & \cdots & \vdots \\ \dfrac{\delta_i^m}{m!} & \dfrac{\delta_i^{m-1}}{(m-1)!} & \cdots & \delta_i\end{bmatrix}$$

(Anderson (1987)).

The algorithm for the vector smoothing polynomial spline can easily be modified if there are missing data (Jones (1984)). For example, suppose y_{1i} is missing for some i. For this case, we would eliminate the first row of the $\mathbf{H}$ matrix of (4) and the first row and column of the observational error covaraince matrix $\mathbf{R}$. A similar modification to these four quantities would be used if y_{2i} was missing.

Given the information above, we can now follow the nine steps of the Kalman filter stated in § 3. In addition, we can also follow the algorithm for the Rauch-Tung-Striebel fixed interval smoothing. As in the univariate case, we assume a diffuse prior, that is, $\mathbf{x}(0|0)\sim\mathbf{N}(\mathbf{0}, \xi\mathbf{I}_{2m})$, $\xi\to\infty$, where $\mathbf{I}_{2m}$ is the $2m\times 2m$ identity matrix. Thus the assumption of a diffuse prior requires that a modification be made for the first m steps of the Kalman filter and the last m steps of the RTS smoothing algorithm (Kohn and Ansley (1987)). The modifications of the filtering and smoothing algorithms have been worked out for the case of multiple responses. A general discussion of the filtering and smoothing modifications is given in Appendix A. We implemented these algorithms using FORTRAN 77. A nonlinear optimization program written by Dennis and Schnabel (1983) was also used to find the maximum likelihood estimates of the parameters, as was described earlier.

In the two-dimensional vector extension of the smoothing polynomial spline, we have a choice of four possible models:

(1) An additive model where there is no correlation in either the plant noise or observation noise of the two splines;

(2) A model that has correlation in the plant noise but not the observational noise;

(3) A model that has correlation in the observational noise but not the plant noise; and

(4) A model that has correlation in both the plant noise and observational noise. In model (1), it is necessary to estimate four parameters. In models (2) and (3), we need to estimate five parameters, and in model (4), we need to estimate six parameters. The strategy we used to find the "best model" for a two-dimensional vector spline of degree $2m-1$ was outlined by Jones (1984). We began by fitting two splines univariately, which is equivalent to model (1). In this paper, we fit the univariate models with the constraint that both splines were of the same degree. After the "best models" were found, we use the estimates, $\hat{\sigma}_{11}$, $\hat{\sigma}_{22}$, $\hat{R}_{11}$, and $\hat{R}_{22}$ as initial estimates for models (2)-(4). The additional parameters in models (2)-(4) had initial guesses of 0.0. The significance of the additional p $(p=1,2)$ parameters was determined by examining the change in $-2\ln$ likelihood due to the addition. This change is distributed asymptotically as χ^2_p. The "best" overall model was obtained by applying *Akaike's Information Criterion* (AIC) (Akaike (1973)),

$$\text{AIC} = -2\ln \text{likelihood} + 2p \tag{19}$$

to the competing models. As in Jones (1984), we considered any model within two units of the minimum AIC to be a competitor for the best overall model.

6. Examinations of assumptions for vector models. Jones (1984) considered a large variety of multivariate time series models. He examined models of the general form:

$$\frac{d}{dt}\begin{bmatrix}\mathbf{x}(t)\\ \mu\end{bmatrix} = \begin{bmatrix}\mathbf{A} & \mathbf{0}\\ 0 & 0\end{bmatrix}\begin{bmatrix}\mathbf{x}(t)\\ \mu\end{bmatrix} + \begin{bmatrix}\mathbf{G}\boldsymbol{\varepsilon}(t)\\ 0\end{bmatrix}, \tag{20}$$

where $\mathbf{x}(t)$ is a $d\times 1$ state vector of deviations from a mean value, μ. In his formulation, $\mathbf{A}$ is the $d\times d$ transition matrix, $\mathbf{G}$ is a $d\times p$ $(p\leqq d)$ matrix defining the structure of the random inputs, and $\boldsymbol{\varepsilon}(t)$ is the vector of random inputs. The solution of the homogeneous part of this equation given the data is

$$\mathbf{x}(t_i) = \mathbf{\Phi}(\delta_i)\mathbf{x}(t_{i-1}), \tag{21}$$

where $\delta_i = t_i - t_{i-1}$, and $\mathbf{\Phi}(\delta_i) = e^{\mathbf{A}\delta_i}$. In the general case,

$$\mathbf{\Phi}(\delta_i) = \mathbf{I} + \sum_{i=1}^{\infty} \frac{(\mathbf{A}\delta_i)^i}{i!}. \tag{22}$$

The problem of representing this infinite sum can be averted if $\mathbf{A}$ is factored in the form:

$$\mathbf{A} = \mathbf{H}\mathbf{\Lambda}\mathbf{H}^{-1}, \tag{23}$$

where $\mathbf{\Lambda}$ is a diagonal matrix of the eigenvalues of $\mathbf{A}$ and $\mathbf{H}$ has columns consisting of the right eigenvectors of $\mathbf{A}$. These eigenvalues and eigenvectors are in the complex domain. Jones then introduced a vector, $\mathbf{Y}(t)$, called the *transformed state*, of the form $\mathbf{Y}(t)=\mathbf{H}^{-1}\mathbf{x}(t)$. He then represented the upper portion of (20) as

$$\frac{d}{dt}\mathbf{Y}(t)=\mathbf{\Lambda}\mathbf{Y}(t)+\mathbf{H}^{-1}\mathbf{G}\boldsymbol{\varepsilon}(\mathrm{t}). \tag{24}$$

The solution of the homogeneous part of (13) can be written as

$$\mathbf{\Phi}(\delta_i)=\begin{bmatrix} e^{\lambda_1} & 0 & \cdots & 0 \\ 0 & e^{\lambda_2} & \cdots & 0 \\ \vdots & \cdots & \ddots & \vdots \\ 0 & 0 & \cdots & e^{\lambda_d} \end{bmatrix}, \tag{25}$$

where the λ_i are the eigenvalues of $\mathbf{A}$. The observation equation was written as

$$\mathbf{Z}(t_i)=\mathbf{H}\mathbf{Y}(t_i)+\mu+V(t_i). \tag{26}$$

The parameters of interest were estimated by using the Kalman filter to calculate $-2\ln$ likelihood and a nonlinear optimization routine to find its minimum.

In our formulation of the bivariate response model, we assume that correlation exists in the plant and/or observation error structures. We assume that the transition matrix has a block diagonal or "uncoupled" structure. Also, in our formulation, we can represent $\mathbf{\Phi}(\delta_i)$ as a finite power series. Furthermore, we assume that $\mathbf{x_B}(t_i|t_i)$ has a distribution that is approximately $\mathrm{N}(\mathbf{x_B}(t_i), \mathbf{P}(t_i))$. Thus, we can obtain approximate " one at a time" and simultaneous confidence intervals. The meaning and statistical properties of confidence intervals in this nonparametric setting depend on the truth of the model assumptions being made (Nychka (1988)). The "one at a time" (Graybill (1976), p. 195) confidence interval for $\mathbf{L_k}^T\mathbf{x_B}(t_i)$ where $\mathbf{L_k}$ is a $2m\times 1$ vector of known constants is obtained by

$$\mathbf{L_k}^T\mathbf{x_B}(t_i|t_i)\mp t_{n-p,1-(\alpha/2)}\sqrt{\mathbf{L_k}^T\mathbf{P}(t_i|t_i)\mathbf{L_k}}. \tag{27}$$

Simultaneous or Scheffe confidence intervals on $\mathbf{L}^T\mathbf{H}\mathbf{x_B}(t_i)$ where $\mathbf{L}$ is any $q\times 1$ vector, H is a $q\times 2m$ matrix of constants of rank q are obtained by the expression:

$$\mathbf{L}^T\mathbf{H}\mathbf{x}(t_i)\mp\sqrt{qF_{q,n-2m,1-\alpha}}\sqrt{\mathbf{L}^T\mathbf{H}\mathbf{P}(t_i|t_i)\mathbf{H}^T\mathbf{L}}. \tag{28}$$

In our case, we let $\mathbf{H}=\mathbf{I_{2m}}$ so that $q=2m$. Also, we can use the asymptotic properties of the F distribution to replace $qF_{q,n-2m,1-\alpha}$ by $\chi^2_{q,1-\alpha}$.

In addition to the confidence intervals of linear combinations of the estimated state vector, we can obtain confidence intervals of a nonlinear form that has a particularly important application. For example, when modeling gas exchange data, we obtain estimates of $\dot{V}_{O_2}$ and $\dot{V}_{CO_2}$ with respect to time. Also, we want to obtain estimates of $d\dot{V}_{O_2}/\mathrm{dt}$ and $d\dot{V}_{CO_2}/\mathrm{dt}$. However, the quantity of interest may be

$$\frac{d\dot{V}_{CO_2}}{d\dot{V}_{O_2}}=\frac{d\dot{V}_{CO_2}/dt}{d\dot{V}_{O_2}/dt}. \tag{29}$$

The approximate confidence interval for the quantity on the left hand side of (29) may be obtained by the use of Fieller's theorem (Fieller (1944) and Zerbe (1978)). For example, suppose that we find the best vector smoothing spline according to our model assumptions has an estimated state vector of the form

$$\mathbf{x_B}(t_i|t_i)=\begin{bmatrix} x_1(t_i|t_i) \\ x_1'(t_i|t_i) \\ x_2(t_i|t_i) \\ x_2'(t_i|t_i) \end{bmatrix}. \tag{30}$$

The $'$ refers to the first derivative taken with respect to time. For our application, we want the confidence interval for

$$\rho=\frac{(0\ 0\ 0\ 1)\mathbf{x}_{\mathbf{B}}(t_i)}{(0\ 1\ 0\ 0)\mathbf{x}_{\mathbf{B}}(t_i)}.$$

For convenience, let $\mathbf{L}_1=(0\ 0\ 0\ 1)$ and $\mathbf{L}_2=(0\ 1\ 0\ 0)$. Following Zerbe (1978), we define

$$\begin{aligned}T=(\mathbf{T}_1^T\mathbf{x}(t_i\,|\,t_i)-\rho\mathbf{L}_2^T\mathbf{x}(t_i\,|\,t_i))/[\mathbf{L}_1^T\mathbf{P}(t_i\,|\,t_i)\mathbf{L}_1\\-2\rho\mathbf{L}_1^T\mathbf{P}(t_i\,|\,t_i)\mathbf{L}_2+\rho^2\mathbf{L}_2^T\mathbf{P}(t_i\,|\,t_i)\mathbf{L}_2]^{1/2}.\end{aligned}\tag{31}$$

Assuming that $\mathbf{e}\sim\mathbf{N}(\mathbf{0},\mathbf{I}\sigma^2)$, then T has a distribution that is a Student's t distribution with $n-p$ degrees of freedom. Thus, it is assumed

$$1-\alpha=\Pr(-t\leqq T\leqq t)=\Pr(A\rho^2+B\rho+C\leqq 0),\tag{32}$$

where

$$A=(\mathbf{L}_2^T\mathbf{x}(t_i\,|\,t_i))^2-t^2\mathbf{L}_2^T\mathbf{P}(t_i\,|\,t_i)\mathbf{L}_2,$$

and

$$B=2\cdot[t^2\mathbf{L}_1^T\mathbf{P}(t_i\,|\,t_i)\mathbf{L}_2-\mathbf{L}_1^T\mathbf{x}(t_i\,|\,t_i)\mathbf{L}_2^T\mathbf{x}(t_i\,|\,t_i)],$$
$$C=(\mathbf{L}_1^T\mathbf{x}(t_i\,|\,t_i))^2-t^2\mathbf{L}_1^T\mathbf{P}(t_i\,|\,t_i)\mathbf{L}_1.$$

In our application, since there is more than one estimated variance parameter, we replace $t^2=F_{1,n-1}$ by u^2/q, where u^2 is asymptotically a chi square distribution with p degrees of freedom (Carter, Wampler, and Stablein, 1983). We can use this asymptotic property because of our large sample sizes. If a, b, and c are the observed values of the random variables A, B, and C, then the confidence interval for ρ is given by

$$\frac{-b\mp(b^2-4ac)^{1/2}}{2a}\tag{33}$$

provided that the discriminant $b^2-4ac\geqq 0$. The Fieller confidence limits become infinite when the denominator and numerator are bivariately not significantly different from 0 (Zerbe et al. (1982)). They are exclusive if the numerator is significantly different from 0 but the denominator is not. This caused difficulty in some of our applications because the estimated derivatives, $d\hat{\dot{V}}_{O_2}/dt$ and $d\hat{\dot{V}}_{CO_2}/dt$ were near 0 during some intervals of time. Thus, in some situations, we were unable to obtain confidence intervals for the derivatives of interest.

7. Applications of vector smoothing polynomial splines. In § 4, we fit two univariate cubic smoothing polynomial splines to the gas exchange data of subject B.D. The curves of interest were $\dot{V}_{O_2}$ versus time and $\dot{V}_{CO_2}$ versus time. Both of the fitted splines indicated the existence of a large amount of noise in the data. The Pearson correlation coefficient between the residuals associated with the two splines was .583, which was significantly different from 0 ($p<0.0001$). Thus, we fit a vector smoothing polynomial spline that modeled $\dot{V}_{CO_2}$ and $\dot{V}_{O_2}$ versus time. Four cubic vector models were considered. The four models were (1) a model that had no correlation in either the observation or plant noise, (2) a model that correlated the two splines in the observation noise but not the plant noise, (3) a model that correlated the two splines in the plant noise but not the observation noise, and (4) a model that correlated the two splines in both the observation and plant noise. The first model was equivalent to fitting each of the two splines univariately. In this case, the overall value of $-2\ln$ likelihood for the bivariate response model was obtained by summing the values of $-2 \ln$ likelihood associated with the two univariate splines. After finding the optimum splines for each of the four models, we determined the best model by applying Akaike's Information Criterion (AIC) to each of the four models. The comparative results are summarized in Table 2.

The best model was determined to be model 4. The estimates of the observation and plant correlation terms were $\hat{\rho}_O = .560$ and $\hat{\rho}_P = .986$, respectively. For this model, we plotted the predicted $\dot{V}_{CO_2}$ and its "one at a time" confidence interval against the predicted $\dot{V}_{O_2}$ (see Fig. 4). Hence, we could attempt to identify a threshold point using the same measures as recommended by Beaver, Wasserman, and Whipp (1986). Also, we obtained $d\hat{\dot{V}}_{CO_2}/d\hat{\dot{V}}_{O_2}$ for each value of $\hat{\dot{V}}_{O_2}$. The Fieller's confidence interval was not plotted because it was infinite at the points where $\hat{\dot{V}}_{O_2}$ was not significantly different from 0.

TABLE 2
Analysis of cubic two-dimensional spline models for subject B.D.

Model	Obs. noise corr.	Plant noise corr.	No. of parameters	AIC
1	No	No	4	9805.4
2	Yes	No	5	9616.3
3	No	Yes	5	9723.6
4	Yes	Yes	6	9561.7

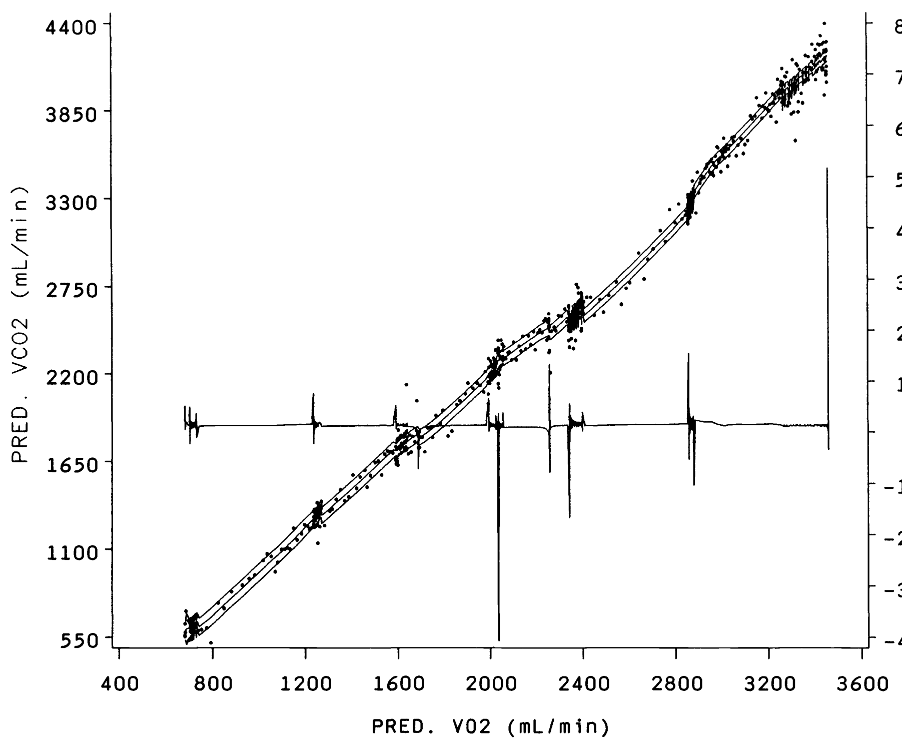


FIG. 4. *Carbon dioxide production plotted against oxygen consumption for subject* B.D. (*increasing curve with confidence intervals*) *based on two-dimensional vector smoothing spline. The slope estimates* (*horizontal line with sharp fluctuations*) *were obtained as the ratio of the two slope estimates.*

Figure 4 reveals that there were several time intervals where the $d\hat{V}_{CO_2}/d\hat{V}_{O_2}$ fluctuated sharply. These periods of rapid fluctuations corresponded roughly to the incremental changes in work load that were introduced to subject B.D. As we stated in § 4, the workload was incremented every three minutes. Thus, the analysis of the vector smoothing spline provided information about the design of the experiment. To identify a possible threshold, we examined the values of the curve where the fluctuations in $d\hat{V}_{CO_2}/d\hat{V}_{O_2}$ were sustained over a period of several breaths. We found that beginning at $\hat{V}_{O_2} = 2857$ mL/min, there was a large fluctuation in $D\hat{V}_{CO_2}/d\hat{V}_{O_2}$ for subject B.D. that lasted about four breaths. These oscillations occurred in the time period from $t = 20.68$ minutes to $t = 20.78$ minutes.

Our second application of the two-dimensional vector smoothing spline was in the analysis of the gas exchange data for subject D.S. To analyze the data bivariately, we once again used a cubic spline and examined four competing models. As for subject B.D., we again found that the best vector cubic smoothing spline was one that had correlation in both the observation noise and the plant noise. The estimates of the observation and plant correlation terms for D.S. were $\hat{\rho}_O = .737$ and $\hat{\rho}_P = .973$, respectively. In this example, we were able to obtain approximate Fieller's confidence intervals for $d\dot{V}_{CO_2}/d\dot{V}_{O_2}$. The best maximum likelihood two-dimensional cubic smoothing spline for D.S. is represented graphically by Fig. 5.

Here we see that $d\hat{V}_{CO_2}/d\hat{V}_{O_2}$ reaches a maximum at $\hat{V}_{O_2} = 2586$ mL/min. At $\hat{V}_{O_2} = 2793$ mL/min, the rate of change of $d\hat{V}_{CO_2}/d\hat{V}_{O_2}$ drops to its lowest point. Outside of the $\hat{V}_{O_2}$ interval (2586, 2793), the rate of change of $d\hat{V}_{CO_2}/d\hat{V}_{O_2}$ is relatively constant.

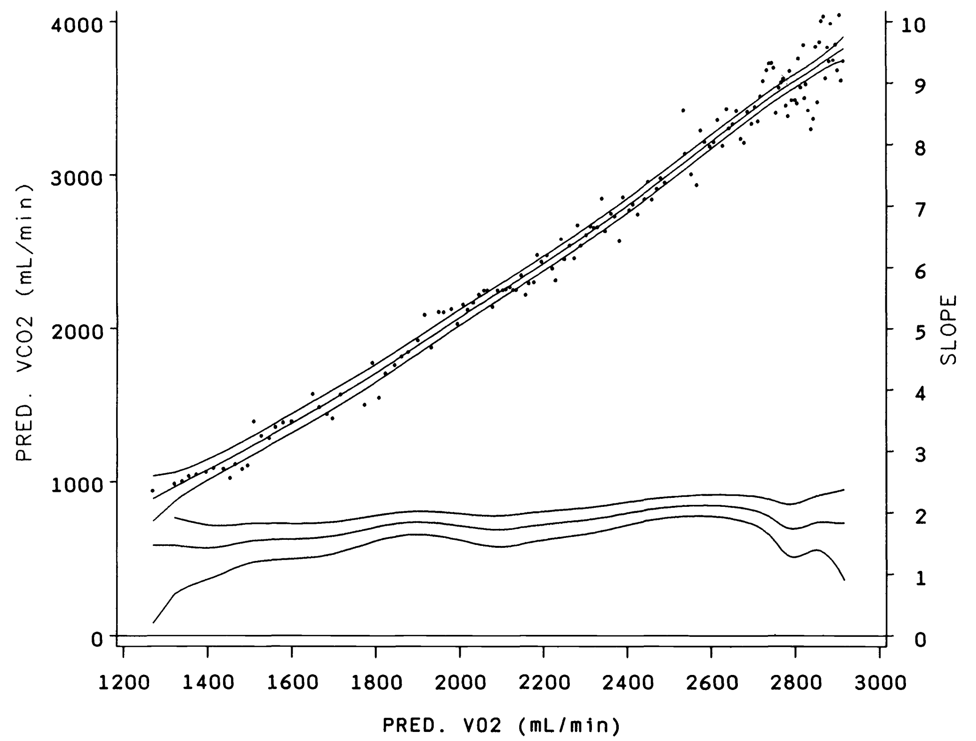


FIG. 5. *Carbon dioxide production plotted against oxygen consumption for subject* D.S. (*increasing curve*) *based on two-dimensional vector smoothing spline. The slope estimates confidence intervals are based on Fieller's theorem.*

We can identify an interval where the gas exchange measurements are somewhat unstable. Thus, the bivariate analysis of the subject's response allows us to extract information about the subject which we were unable to do with separate univariate analyses.

8. Conclusions and directions for further research.

In this paper, we have demonstrated how the state space representation of a smoothing polynomial spline allows us to extend the model so that two correlated processes can be fitted simultaneously. This stochastic motivation of the model provides a natural way to incorporate correlation into the error structure and into the underlying modeled processes. Furthermore, this work can be extended so that an arbitrary number of simultaneous processes can be modeled.

Wahba (1985) compares generalized maximum likelihood (GML) and generalized cross validation (GCV) to estimate the smoothing parameter in a univariate model. In that work, Wahba points out some shortcomings in the GML approach to arriving at a "best" smoothing parameter. For our purposes, however, we found that generalized maximum likelihood provided an easier way to include correlation in our model.

Finally, there is room for much theoretical work to determine what (if any) optimality conditions this model satisfies.

Appendix A. Filtering and smoothing modifications for diffuse priors.

Kohn and Ansley (1985), (1987) and Ansley and Kohn (1985) have developed methods for starting the Kalman filter for nonstationary models using diffuse (noninformative) priors. Here, we derive a start-up method for integrated random walks that is equivalent to Ansley and Kohn's method. The advantage is that the motivation and derivation is much simpler.

For a random walk that is integrated $m-1$ times, the state has m elements, and m observations are necessary before the state can be estimated from the observations. Before any observations are taken, the initial state vector can be taken to be zero with a large covariance matrix,

$$\mathbf{x}(t_1|0)=\mathbf{0}$$

$$\mathbf{P}(t_1|0)=\lambda\mathbf{I},$$

where $\lambda\to\infty$. After one observation, the upper left-hand element becomes finite while the lower right-hand block stays infinite. As each new observation becomes available, the size of the finite upper left-hand block increases by one, and the size of the infinite block decreases by one; therefore, an inductive procedure can be derived.

To begin the procedure, the innovation at the first time point is the first observation,

$$\mathbf{I}(t_1)=y(t_1),$$

with variance

$$V(t_1)=\lambda+R.$$

This variance becomes infinite as $\lambda\to\infty$, so the contribution to $-2\ln$ likelihood is zero. The Kalman gain vector is

$$\mathbf{K}(t_1)=\frac{1}{\lambda+R}\begin{bmatrix}\lambda\\0\\\vdots\\0\end{bmatrix}.$$

The first element approaches one, giving all the weight to the most recent observation. The updated estimate of the state vector is

$$\mathbf{x}(t_1 | t_1) = \begin{bmatrix} y(t_1) \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

with covariance matrix

$$\mathbf{P}(t_1 | t_1) = \lambda \mathbf{I} - \frac{1}{\lambda + R} \begin{bmatrix} \lambda \\ 0 \\ \vdots \\ 0 \end{bmatrix} [\lambda \quad 0 \quad \cdots \quad 0].$$

The upper left-hand corner of this matrix is

$$\mathbf{P}_{11}(t_1 | t_1) = \lambda - \frac{\lambda^2}{\lambda + R} = \frac{\lambda R}{\lambda + R} \xrightarrow{\lambda \to \infty} R.$$

It is important not to take the limit before the subtraction because of the cancellation of the λ^2 terms. Now,

$$\mathbf{P}(t_1 | t_1) = \left[\begin{array}{c:c} R & 0 \\ \hdashline 0 & \lambda \mathbf{I} \end{array}\right].$$

The general step of the starting algorithm works with the upper left-hand corner of $\mathbf{P}(t_i | t_i)$ for $1 \leqq i < m$. This matrix is i by i, and at the completion of the step is augmented to $i+1$ by $i+1$. The modified recursion follows:

(1) The one-step prediction is calculated as usual, except that only the first k elements need be forecast, and the predicted state vector augmented with zeros,

$$\mathbf{x}(t_k | t_{k-1}) = \mathbf{\Phi}(\delta_k) \mathbf{x}(t_k | t_{k-1})$$

(2) The covariance matrix of the one-step prediction approaches ∞ as $\lambda \to \infty$ since the large lower right-hand corner block is propagated into the entire matrix by the state transition matrix. This has implications in the smoothing step to come later,

$$\mathbf{P}^{(0)}(t_{i+1} | t_i) + \lambda \mathbf{\Delta}_i \mathbf{\Delta}_i^T,$$

where

$$\mathbf{\Delta}_i = \begin{bmatrix} \delta_i^i \\ \delta_i^{i-1} \\ \vdots \\ 1 \end{bmatrix},$$

and $\mathbf{P}^{(0)}(t_{i+1} | t_i)$ is the matrix calculated by the usual method with a 0 in the lower right-hand corner of $\mathbf{P}(t_i | t_i)$ instead of a λ.

(3) Since

$$\mathbf{H} \mathbf{\Delta}_i = \delta_i^i,$$

the innovation variance becomes

$$V(t_i) = V^{(0)}(t_i) + \lambda \delta_i^{2i}$$

where

$$V^{(0)}(t_i)=P_{11}^{(0)}(t_{i+1}\,|\,t_i)+R.$$

(4) The Kalman gain is

$$\mathbf{K}(t_i)=\frac{1}{V^{(0)}+\lambda\delta_i^{2i}}[\mathbf{P}^{(0)}(t_i\,|\,t_{i-1})\mathbf{H}^T+\lambda\boldsymbol{\Delta}_i\delta_i^i]$$

and the updated state vector, for large λ approaches

$$\mathbf{x}(t_i\,|\,t_i)=\mathbf{x}(t_i\,|\,t_{i-1})+\frac{1}{\delta_i^i}\boldsymbol{\Delta}_i I(t_i).$$

(5) The updated state covariance matrix is

$$\mathbf{P}^{(0)}(t_i\,|\,t_{i-1})+\lambda\boldsymbol{\Delta}_i\boldsymbol{\Delta}_i^T-\mathbf{K}(t_i)\mathbf{H}[\mathbf{P}^{(0)}(t_i\,|\,t_{i-1})+\lambda\boldsymbol{\Delta}_i\boldsymbol{\Delta}_i^T].$$

Putting all terms over the common denominator $V^{(0)}(t_i)+\lambda\delta_i^{2i}$, the terms in the numerator containing λ^2 cancel. Keeping only the terms in the numerator and denominator that are multiplied by λ gives in the limit, for large λ,

$$\mathbf{P}^{(0)}(t_i\,|\,t_i)=\mathbf{P}^{(0)}(t_i\,|\,t_{i-1})+\frac{V^{(0)}}{\delta_i^{2i}}\boldsymbol{\Delta}_i\boldsymbol{\Delta}_i^T-\frac{1}{\delta_i^i}[\mathbf{P}^{(0)}(t_i\,|\,t_{i-1})\mathbf{H}^T\boldsymbol{\Delta}_i^T+\boldsymbol{\Delta}_i\mathbf{H}\mathbf{P}^{(0)}(t_i\,|\,t_{i-1})].$$

This completes the modification of the first m steps of the Kalman filter in the univariate case. The smoothing is carried out as before from time t_n to t_m. The values for the first $m-1$ time points are simply backward predictions starting at time m,

$$\mathbf{x}(t_i\,|\,t_n)=\boldsymbol{\Phi}(-\delta_{i+1})\mathbf{x}(t_{i+1}\,|\,t_n)$$
$$\mathbf{P}(t_i\,|\,t_n)=\boldsymbol{\Phi}(-\delta_{i+1})\mathbf{P}(t_{i+1}\,|\,t_n)\boldsymbol{\Phi}^T(-\delta_{i+1})-\mathbf{Q}(-\delta_{i+1}).$$

In the two-dimensional case, the upper and lower halves of the state vector and the upper left-hand and lower right-hand blocks of the state covariance matrix are built up as in the univariate case. The off-diagonal blocks remain zero until time point m. Even though $\mathbf{Q}$ and $\mathbf{R}$ introduce off-diagonal elements in the covariance matrix, these are dominated by λ until the state is determined at time t_m.

Acknowledgments. The authors would like to thank the referees for their very useful comments.

REFERENCES

H. AKAIKE (1973), *Information theory and an extension of the maximum likelihood principle*, Second Internat. Symposium on Information Theory, B. N. Petrov and F. Csaki, eds., Akademia Kaido, Budapest, pp. 267-281.

S. J. ANDERSON (1987), *Smoothing polynomial splines with applications to modeling univariate, bivariate, and growth curve data*, Ph.D. thesis, Graduate Program in Biometrics, School of Medicine, University of Colorado Health Sciences Center, Denver, CO.

C. F. ANSLEY AND R. KOHN (1987), *Efficient generalized cross-validation for state space models*, Biometrika, 74, pp. 139-148.

———, (1985), *Estimation, filtering and smoothing in state space models, with incompletely specified initial conditions*, Ann. Statist., 13, pp. 1286-1316.

W. L. BEAVER, K. WASSERMAN, AND B. J. WHIPP, (1986), *A new method for detecting anaerobic threshold by gas exchange*, J. Appl. Phys., 60(6), pp. 2020-2027.

R. G. BROWN (1983), *Introduction to Random Signal Analysis and Kalman Filtering*, John Wiley, New York.

W. H. CARTER, G. L. WAMPLER, AND D. M. STABLEIN (1983), *Regression Analysis of Survival Data in Cancer Chemotherapy*, Marcel Dekker, New York, pp. 13-22.

P. CRAVEN AND G. WAHBA (1979), *Smoothing noisy data with spline functions*, Numer. Math., 31, pp. 377-403.

J. E. DENNIS AND R. B. SCHNABEL (1983), *Nonlinear Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ.

E. C. Fieller (1944), *A fundamental formula in the statistics of biological assay, and some applications*, Quart. J. of Pharmacy and Pharmacology, 17, pp. 117-123.

F. A. Graybill (1976), *Theory and Application of the Linear Model*, Duxbury Press, North Scituate, MA.

M. F. Hutchinson and F. R. De Hoog (1985), *Smoothing noisy data with spline functions*, Numer. Math., 47, pp. 107-122.

R. H. Jones (1984), *Fitting multivariate models to unequally spaced data*, Time Series Analysis of Irregularly Observed Data, E. Parzen, ed., Lecture Notes in Statistics, 25, Springer-Verlag, Berlin, NY, pp. 158-188.

R. H. Jones and P. V. Tyron (1987), *Continuous time series models for unequally spaced data applied to modeling atomic clocks*, SIAM J. Sci. Statist. Comput., 8 pp. 71-81.

P. de Jong (1988), *The likelihood for a state space model*, Biometrika, 75, pp. 165-169.

R. E. Kalman (1960), *A new approach to linear filtering and prediction problems*, Trans. ASME J. Basic Engrg. 82D, pp. 35-45.

G. S. Kimeldorf and G. Wahba (1970), *A correspondence between Bayesian estimation on stochastic processes and smoothing by splines*, Ann. Math. Statist., 41, pp. 495-502.

R. Kohn and C. F. Ansley, (1985), *A structured state space approach to computing the likelihood and its derivates*, J. Statist. Comput. Simulation, 21, pp. 135-169.

———, (1987), *A new algorithm for spline smoothing based on smoothing a stochastic process*, SIAM J. Sci. Statist. Comput., 8, pp. 33-48.

D. Nychka (1988), *Bayesian confidence intervals for smoothing splines*, J. Amer. Statist. Assoc., 83, pp. 1134-1143.

H. E. Rauch, F. Tung, and C. T. Striebel (1965), *Maximum likelihood estimates of linear dynamic systems*, AIAA J., 8, pp. 1445-1450.

A. P. Sage and J. L. Melsa (1971), *Estimation Theory with Applications To Communications and Control*, McGraw-Hill, New York, pp. 347-376.

I. J. Schoenberg (1964), *Spline functions and the problem of graduation*, in Proc. of the National Academy of Sciences, 52, pp. 947-950.

F. C. Schweppe (1965), *Evaluation of likelihood functions for gaussian signals*, IEEE Trans. Inform. Theory, 11, pp. 61-70.

D. L. Sherrill (1987), *On the modeling of respiratory response data*, Ph.D. thesis, Graduate Program in Biometrics, School of Medicine, University of Colorado Health Sciences Center, Denver, CO.

T. N. Thiele (1980), *Om anvedelse af mindste kvadraters methode i nogle tilfaelde, hvor en komplikation af visse slags uensartede tilfaeldige fejlkilder giver fejlene en "systematisk" karakter*, Vidensk. Selsk. French translation: *Sur la compensation de quelques erreurs quasisystematiques par la méthode des moindres carreés*, Kobenhaven, Rietzel, 1880. Vidensk. Selsk. Skr. 5. rk., Naturvid og Mat. Afd., 12, pp. 381-408.

T. D. Wade, S. J. Anderson, J. Bondy, V. A. Ramadevi, R. H. Jones, and G. W. Swanson (1988), *Using smoothing splines to make inferences about the shape of gas-exchange curves*, Comput. Biomed. Res., 21, pp. 16-26.

G. Wahba (1978), *Improper priors, spline smoothing and the problem of guarding against model errors in regression*, J. Royal Statist. Soc. Ser. B, 40, pp. 364-372.

———, (1985), *A comparison of* GCV *and* GML *for choosing the smoothing parameter in the generalized spline smoothing problem*, Ann. Statist., 13, pp. 1378-1402.

W. Wecker and C. F. Ansley (1983), *The signal extraction approach to nonlinear regression and spline smoothing*, J. Amer. Statist. Assoc., 78, pp. 81-89.

H. L. Weinert, R. H. Byrd, and G. S. Sidhu (1980), *A stochastic framework for recursive computation of spline functions*: *Part* II, *Smoothing splines*, J. Optim. Theory and Appl., 30, pp. 255-268.

S. Wold (1974), *Spline functions in data analysis*, Technometrics, 16, pp. 1-11.

G. O. Zerbe (1978), *On Fieller's theorem and the general linear model*, Amer. Statist., 32, pp. 103-105.

G. O. Zerbe, E. Laska, M. Meisner, and H. B. Kushner, (1982), *On multivariate confidence regions and simultaneous confidence limits for ratios*, Comm. Statist., 11(21), pp. 2401-2425.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 4, pp. 767-793, July 1990

TWO-COLOR FOURIER ANALYSIS OF ITERATIVE ALGORITHMS FOR ELLIPTIC PROBLEMS WITH RED/BLACK ORDERING[*]

C.-C. JAY KUO[†] AND TONY F. CHAN[‡]

Abstract. The red/black ordering scheme is often used to increase the parallelism of iterative methods for solving elliptic partial differential equations (PDEs). However, the convergence rates are also affected, often adversely. This paper provides a unified approach, called the two-color Fourier analysis, to study the convergence rates of iterative algorithms for elliptic problems with the red/black ordering. This Fourier tool is used to analyze different types of iterative algorithms, including the successive over-relaxation (SOR) method, symmetric successive over-relaxation (SSOR) method, preconditioned iterative methods with SSOR, ILU, and MILU preconditioners, and multigrid (MG) methods. By comparing the convergence rates of algorithms with the natural and red/black orderings, it is shown that although the red/black ordering does not affect the rate of convergence in the context of SOR and MG methods, it slows down the convergence significantly in the context of SSOR and preconditioned iterative methods.

Key words. Fourier analysis, incomplete factorization, multigrid method, parallel computation, preconditioned conjugate gradient, preconditioners, red/black ordering, successive over-relaxation, symmetric successive over-relaxation

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

1. Introduction.

An important task of the research on parallel computation is to seek algorithms that can be conveniently implemented on vector or parallel computers. One common approach to obtain parallel iterative algorithms for the solution of partial differential equations (PDEs) is *reordering*. By reordering, we rearrange the computational sequences to increase the percentage of computations that can be done independently [27]. A crucial issue associated with reordering is how the convergence rate of an iterative algorithm is affected by a reordering scheme.

The multicolor ordering scheme for grid points provides more parallelism than the natural rowwise or columnwise ordering scheme. It is well known that by using red and black, two colors to order the grid points in a checkerboard fashion for the 5-point Laplacian, we are able to separate the coupling between any two red (or black) points so that the values at all red (or black) points can be updated simultaneously. Similarly, four colors are needed to separate the coupling between grid points of the same color for the 9-point Laplacian [1]-[4],[20],[22],[23]. On either vector or parallel computers, an algorithm with the multicolor ordering is always easier to vectorize or parallelize than its naturally ordered counterpart so that such a reordering is attractive for parallel implementation. There are numerous discussions on the implementation of iterative algorithms with the red/black ordering on vector and parallel computers in the literature, for example, in [1],[5],[8],[11],[23],[27],[28], and [33].

In this paper, we examine how the convergence rate of an iterative algorithm is affected by the red/black ordering. Our study includes the successive over-relaxation (SOR), symmetric successive over-relaxation (SSOR), ILU, and MILU preconditioners

[*] Received by the editors June 6, 1988; accepted for publication (in revised form) April 26, 1989. This work was partially supported by U.S. Department of Energy contract DE-FG03-87ER25037, National Science Foundation contract NSF-DMS87-14612, and U.S. Army Research Office contract DAAL03-88-K-0085.

[†] Signal and Image Processing Institute and Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, California 90089-0272.

[‡] Department of Mathematics, University of California, 405 Hilgard Ave., Los Angeles, California 90024.

for preconditioned iterative methods, and multigrid (MG) methods. The convergence rates of these algorithms are analyzed by a unified approach called the *two-color Fourier analysis.* Although the two-color Fourier analysis has been used in analyzing the SOR and MG methods by Kuo, Levy, and Musicus [20],[21],[23], we believe that results for the SSOR iteration and the SSOR, ILU, MILU preconditioners are new.

Fourier or modified Fourier analysis has been used successfully to analyze numerical methods for elliptic PDE problems for years. We can conveniently study the effects of operators on Fourier modes if the numerical method of interest is applied to a simple model problem that consists of a constant-coefficient PDE on a regular domain with appropriate boundary conditions. The model problem for second-order self-adjoint elliptic PDEs is the Poisson equation on a square with Dirichlet boundary conditions. For the model Poisson problem, the SOR iteration was analyzed with Fourier-like basis functions by Frankel [18] and Young [30]. Brandt used Fourier analysis to study the error smoothing property for multigrid methods [10]. Stüben and Trottenberg performed a two-grid analysis to analyze both the error smoothing and the coarse-grid correction with Fourier basis functions [29]. Fourier analysis has also been applied to the analysis of the 5-point or 9-point SOR iteration with the natural or multicolor ordering [3],[20],[22]-[24], preconditioners for elliptic problems with the natural ordering [13], and problems arising from the domain decomposition context [12],[14].

Due to the multicolor ordering scheme, the resulting system of iteration equations is not spatially homogeneous but is *periodic* with respect to grid points. Consequently, the Fourier modes are not eigenfunctions for the multicolor system, and therefore a straightforward Fourier analysis does not apply. When these Fourier modes are operated by periodic operators, there exists a coupling between high and low frequency components. By exploiting the periodic property, we reformulate the conventional Fourier analysis as a two-color Fourier analysis. From this new viewpoint, components in the high frequency region are folded into the low frequency region so that there exist two, i.e., red and black, computational waves in the low frequency region. The coupling between the low and high conventional Fourier components is therefore transformed into a coupling between the red and black computational waves with the same frequency in the low frequency region. With this new Fourier tool, the spectral representation of operators with the red/black ordering can be easily derived and interpreted. For the model Poisson problem, the two-color Fourier analysis is exact for Dirichlet boundary conditions and, with some modifications, is also applicable to periodic boundary conditions. The two-color Fourier analysis can be generalized to the *multicolor* Fourier analysis, which applies to ordering schemes with more than two colors [22].

The determination of the optimal relaxation parameters of the SOR method with the multicolor ordering and their corresponding convergence rates for both 5-point and 9-point Laplacian operators have been intensively investigated [3],[22]-[24]. It has been found that if the relaxation parameters are appropriately selected, the numbers of iterations required for the red/black and natural orderings should be of the same order. In the context of MG methods, the red/black Gauss-Seidel smoother provides a better smoothing rate than the lexicographical Gauss-Seidel smoother [29]. Hence, the red/black reordering does not deteriorate the performance for these two types of algorithms.

However, the same conclusion does not apply to the SSOR iteration and preconditioned iterative methods. The optimal relaxation parameter and its corresponding convergence rate of the SSOR iteration depends highly on the ordering [7],[19],[32]. The naturally ordered SSOR method has the same order of convergence rate as the SOR method and can be accelerated to give an even faster convergence rate by the Chebyshev semi-iterative or conjugate gradient procedure [9],[19],[32]. In contrast, for the red/black ordering, it has been observed that the optimal relaxation parameter for the SSOR method is 1 so that the resulting scheme reduces to a forward and backward Gauss-Seidel relaxation which converges much slower [19]. Here, we use the two-color Fourier analysis to analyze the red/black SSOR method and determine its optimal relaxation parameter 1 analytically. We also perform a quantitative study of the eigenstructure of the preconditioned Laplacian operator with the SSOR, ILU, and MILU preconditioners. The results indicate that the condition number of the preconditioned operator with the red/black ordering is, in general, one order higher than that of its naturally ordered counterpart. Hence, for SSOR and preconditioned iterative methods, the convergence rate is greatly sacrificed in order to obtain more parallelism.

This paper is organized as follows. The two-color Fourier analytical approach is described and the model problem is formulated accordingly in §2. Section 3 analyzes the convergence rates of the SOR and SSOR iterations. Section 4 studies the eigenstructure of the preconditioned Laplacian operator with the SSOR, ILU, and MILU preconditioners. Then, we perform a two-grid analysis to understand the convergence behavior of the multigrid method in §5. Section 6 compares the convergence rates of iterative algorithms with natural and red/black orderings. Related research work and extensions are given in §§7 and 8.

2. Preliminaries.

2.1. Two-color Fourier analysis.

Consider a two-dimensional sequence $u_{j,k}$ defined on a grid

$$\Omega_h = \{(jh, kh) : 0 \le j, k \le M,\ \ M = h^{-1} \text{ even}\} \tag{2.1}$$

with zero boundary values, i.e., $u_{j,k} = 0$ if $j, k = 0$ or M. We can expand it with Fourier series as

$$u_{j,k} = \sum_{\xi=1}^{M-1} \sum_{\eta=1}^{M-1} \hat{u}_{\xi,\eta} \sin(\xi\pi jh) \sin(\eta\pi kh). \tag{2.2}$$

As usual we call the grid point with index (j, k) the *red* or *black* point, depending on whether $j + k$ is even or odd. The function $u_{j,k}$ at the red and black points defines two sequences: the red sequence $u_{r,j,k}$ and the black sequence $u_{b,j,k}$. They can be expanded in Fourier series, respectively, as

$$u_{r,j,k} = \sum_{(\xi,\eta)\in K_r} \hat{u}_{r,\xi,\eta} \sin(\xi\pi jh) \sin(\eta\pi kh), \quad j + k \text{ even}, \tag{2.3a}$$

$$u_{b,j,k} = \sum_{(\xi,\eta)\in K_b} \hat{u}_{b,\xi,\eta} \sin(\xi\pi jh) \sin(\eta\pi kh), \quad j + k \text{ odd}, \tag{2.3b}$$

where

$$K_b = K = \{(\xi, \eta) \in I^2 : \xi + \eta \le M - 1,\ \xi, \eta \ge 1 \text{ or } \eta = M - \xi,\ 1 \le \xi \le \frac{M}{2} - 1\},$$

and

$$K_r = K \cup \left\{ \left(\frac{M}{2}, \frac{M}{2} \right) \right\}.$$

It is straightforward to check that the Fourier coefficients $\hat{u}_{\xi,\eta}$, $\hat{u}_{M-\xi,M-\eta}$ in (2.2) and $\hat{u}_{r,\xi,\eta}$, $\hat{u}_{b,\xi,\eta}$ in (2.3) are related via

$$\text{(2.4a)} \qquad \begin{pmatrix} \hat{u}_{r,\xi,\eta} \\ \hat{u}_{b,\xi,\eta} \end{pmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{pmatrix} \hat{u}_{\xi,\eta} \\ \hat{u}_{M-\xi,M-\eta} \end{pmatrix}, \quad (\xi,\eta) \in K,$$

$$\text{(2.4b)} \qquad \hat{u}_{r,\xi,\eta} = \hat{u}_{\xi,\eta}, \quad (\xi,\eta) = \left(\frac{M}{2}, \frac{M}{2} \right).$$

We can interpret (2.4) as follows. Through the red/black decomposition (2.3), the component $(M-\xi, M-\eta)$ in the high frequency region is folded into the component (ξ,η) in the low frequency region so that there exist two computational waves in the low frequency region. The original and the folded two-color Fourier domains are depicted in Fig. 1. Note also that K_r and K_b differ only by a single element $(M/2, M/2)$ and, therefore, at the frequency $(M/2, M/2)$ we have only a scalar $\hat{u}_{r,M/2,M/2}$, which is considered as the degenerate case.

2.2. Model problem: A two-wave formulation. Consider the discretized two-dimensional Poisson equation on the square $[0,1]^2$ with grid spacing h,

$$\text{(2.5)} \quad \frac{1}{h^2}(u_{j-1,k} + u_{j+1,k} + u_{j,k-1} + u_{j,k+1} - 4u_{j,k}) = f_{j,k}, \quad 1 \le j,\ k \le M-1,$$

where $M = h^{-1}$ is even and $u_{j,k}$ is given for $j,k = 0$ or M. Without loss of generality, we only consider the case where $u_{j,k}$ is zero on boundaries, since a nonzero $u_{j,k}$ on the boundary can always be moved to the right-hand side and treated as part of the driving function. In addition, since the driving term $f_{j,k}$ with $j,k = 0$ or M does not appear in (2.5), it can be viewed as zero. Consequently, the red/black Fourier series expansion (2.3) for both $u_{j,k}$ and $f_{j,k}$ is well defined. By substituting (2.3) into (2.5) and relating the Fourier coefficients of red and black waves, we can transform (2.5) from the space domain into the red/black Fourier domain. It is a block diagonal matrix equation, in which the equation for a nondegenerate frequency (ξ,η) can be written as

$$\text{(2.6a)} \qquad \begin{bmatrix} 1 & -\alpha_{\xi,\eta} \\ -\alpha_{\xi,\eta} & 1 \end{bmatrix} \begin{pmatrix} \hat{u}_{r,\xi,\eta} \\ \hat{u}_{b,\xi,\eta} \end{pmatrix} = -\frac{h^2}{4} \begin{pmatrix} \hat{f}_{r,\xi,\eta} \\ \hat{f}_{b,\xi,\eta} \end{pmatrix},$$

where

$$\text{(2.6b)} \qquad \alpha_{\xi,\eta} = \frac{\cos(\xi\pi h) + \cos(\eta\pi h)}{2}.$$

Since $(\xi,\eta) \in K$, $0 \le \alpha_{\xi,\eta} < 1$. Only the nondegenerate case will be considered in this paper, since the degenerate case can be analyzed similarly and, in general, it does not change the conclusion for each case.

We can use the familiar matrix approach to derive the same result. Consider a general coefficient matrix with the red/black ordering expressed in block form:

$$\begin{bmatrix} D_r & -C \\ -C^T & D_b \end{bmatrix}.$$

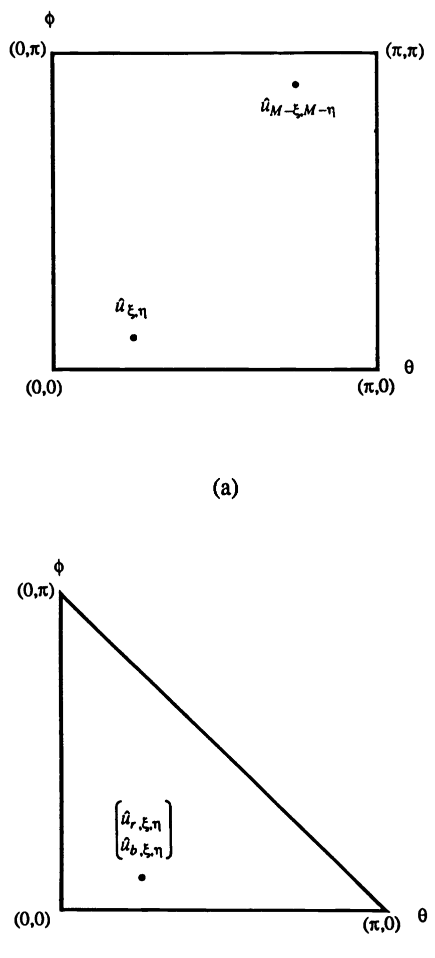


FIG. 1. (a) *Conventional and* (b) *folded two-color Fourier domains, where* $\theta = \xi\pi h$ *and* $\phi = \eta\pi h$.

Let the singular value decomposition for D_r, D_b, and C be

$$D_r = U\hat{D}_r U^T, \quad D_b = V\hat{D}_b V^T, \quad C = U\hat{C}V^T,$$

where matrices U and V (with scaling constants) are defined by the two-color Fourier series expansion (2.3). Then,

$$W = \begin{bmatrix} U & 0 \\ 0 & V \end{bmatrix}$$

is orthogonal and

$$W^T \begin{bmatrix} D_r & -C \\ -C^T & D_b \end{bmatrix} W = \begin{bmatrix} \hat{D}_r & -\hat{C} \\ -\hat{C}^T & \hat{D}_b \end{bmatrix}. \tag{2.7}$$

Since $\hat{D}_r$, $\hat{D}_b$, and $\hat{C}$ are diagonal, (2.7) can be permuted to block diagonal form with 2×2 diagonal blocks. For the system (2.5) scaled by $-h^2/4$, D_r and D_b are identity matrices and the 2×2 diagonal blocks are of the form

$$\begin{bmatrix} 1 & -\alpha_{\xi,\eta} \\ -\alpha_{\xi,\eta} & 1 \end{bmatrix},$$

where $\alpha_{\xi,\eta}$ is defined in (2.6b).

We will use the shift operator notation to represent various operators discussed in this paper, since the conventional matrix notation hides useful geometrical information of variables defined on two-dimensional grids. For example, we express the local Laplacian operator $A_{j,k}$ at grid point (jh, kh) as

$$A_{j,k} = 1 - \frac{E_x + E_x^{-1} + E_y + E_y^{-1}}{4}, \tag{2.8}$$

where E_x and E_y are shift operators along the x- and y- directions. The system (2.5) can therefore be written as

$$A_{j,k} u_{j,k} = -\frac{h^2}{4} f_{j,k}. \tag{2.9}$$

We use A to denote the global operator, which consists of local operators $A_{j,k}$, $1 \leq j, k \leq M-1$ associated with zero boundary values. Besides, $\hat{A}(\xi, \eta)$ is used to denote

$$\hat{A}(\xi,\eta) = \begin{bmatrix} 1 & -\alpha_{\xi,\eta} \\ -\alpha_{\xi,\eta} & 1 \end{bmatrix}, \quad (\xi,\eta) \in K, \tag{2.10}$$

which is the coefficient matrix in the frequency domain as given by (2.6). An equivalent point of view for the global operator A is to treat it as a homogeneous operator defined on an infinite two-dimensional grid and to impose the zero boundary conditions by requiring that input sequences be synthesized with Fourier components given by (2.3) only. By adopting such a viewpoint, the operator algebra [16] can be conveniently applied to manipulate A while its frequency domain expression remains the same.

3. Analysis of SOR and SSOR methods.

3.1. SOR iteration. For the model problem (2.5), the red/black SOR iteration can be written as

$$(3.1)\qquad \begin{aligned} u_{j,k}^{n+\frac{1}{2}} &= S_{r,j,k}(\omega)u_{j,k}^{n} - P_{r,j,k}(\omega)\tfrac{h^2}{4}f_{j,k},\\ u_{j,k}^{n+1} &= S_{b,j,k}(\omega)u_{j,k}^{n+\frac{1}{2}} - P_{b,j,k}(\omega)\tfrac{h^2}{4}f_{j,k}, \end{aligned}$$

where

$$S_{r,j,k}(\omega) = \begin{cases} 1-\omega+\frac{\omega}{4}(E_x+E_x^{-1}+E_y+E_y^{-1}), & (j,k) \text{ red},\\ 1, & (j,k) \text{ black}, \end{cases}$$

$$S_{b,j,k}(\omega) = \begin{cases} 1, & (j,k) \text{ red},\\ 1-\omega+\frac{\omega}{4}(E_x+E_x^{-1}+E_y+E_y^{-1}), & (j,k) \text{ black}, \end{cases}$$

are the local SOR iteration operators at red and black points, and

$$P_{r,j,k}(\omega) = \begin{cases} \omega, & (j,k) \text{ red},\\ 0, & (j,k) \text{ black}, \end{cases} \qquad P_{b,j,k}(\omega) = \begin{cases} 0, & (j,k) \text{ red},\\ \omega, & (j,k) \text{ black}, \end{cases}$$

can be viewed as the local injection operators at red and black points scaled by the parameter ω. As before, we denote their corresponding global operators by S_r, S_b, P_r, and P_b respectively.

By using the red/black Fourier series expansion (2.3), we can transform (3.1) from the space domain to the frequency domain and obtain a block diagonal matrix equation. For each nondegenerate frequency (ξ,η), the iteration equation can be written as

$$(3.2)\quad \begin{aligned} \begin{pmatrix} \hat{u}_{r,\xi,\eta}^{n+1} \\ \hat{u}_{b,\xi,\eta}^{n+1} \end{pmatrix} &= \hat{S}_b(\xi,\eta,\omega)\hat{S}_r(\xi,\eta,\omega)\begin{pmatrix} \hat{u}_{r,\xi,\eta}^{n} \\ \hat{u}_{b,\xi,\eta}^{n} \end{pmatrix}\\ &\quad -\frac{h^2}{4}[\hat{S}_b(\xi,\eta,\omega)\hat{P}_r(\xi,\eta,\omega)+\hat{P}_b(\xi,\eta,\omega)]\begin{pmatrix} \hat{f}_{r,\xi,\eta} \\ \hat{f}_{b,\xi,\eta} \end{pmatrix},\\ &= \begin{bmatrix} 1 & 0 \\ \omega\alpha_{\xi,\eta} & 1-\omega \end{bmatrix}\begin{bmatrix} 1-\omega & \omega\alpha_{\xi,\eta} \\ 0 & 1 \end{bmatrix}\begin{pmatrix} \hat{u}_{r,\xi,\eta}^{n} \\ \hat{u}_{b,\xi,\eta}^{n} \end{pmatrix}\\ &\quad -\frac{h^2}{4}\begin{bmatrix} \omega & 0 \\ \omega^2\alpha_{\xi,\eta} & \omega \end{bmatrix}\begin{pmatrix} \hat{f}_{r,\xi,\eta} \\ \hat{f}_{b,\xi,\eta} \end{pmatrix}, \end{aligned}$$

where $\alpha_{\xi,\eta}$ is given by (2.6b).

For the error, equation (3.2) is a homogeneous equation, and the error dynamic can be completely understood by studying the SOR iteration matrices

$$(3.3)\quad \hat{S}_{rb}(\xi,\eta,\omega) = \hat{S}_b(\xi,\eta,\omega)\hat{S}_r(\xi,\eta,\omega) = \begin{bmatrix} 1-\omega & \omega\alpha_{\xi,\eta} \\ (1-\omega)\omega\alpha_{\xi,\eta} & 1-\omega+\omega^2\alpha_{\xi,\eta}^2 \end{bmatrix}.$$

The objective is to find the optimal relaxation parameter ω^* that minimizes the spectral radius ρ of the matrix S_{rb} with respect to all possible ξ and η and its corresponding spectral radius.

To do so, let us first consider fixed ξ and η. The spectral radius $\rho_{\xi,\eta}(\omega)$ of $\hat{S}_{rb}(\xi,\eta,\omega)$ can be found by solving the quadratic equation

$$|\lambda_{\xi,\eta}(\omega)-\hat{S}_{rb}(\xi,\eta,\omega)| = \lambda_{\xi,\eta}^2-(2-2\omega+\omega^2\alpha_{\xi,\eta}^2)\lambda_{\xi,\eta}+(1-\omega)^2 = 0,$$

so that

$$(3.4)\quad \rho_{\xi,\eta}(\omega) = \max|\lambda_{\xi,\eta}(\omega)| = \begin{cases} \omega - 1, & \omega^*_{\xi,\eta} \le \omega < 2, \\ \left[\frac{\omega\alpha_{\xi,\eta}+[\omega^2\alpha^2_{\xi,\eta}-4(\omega-1)]^{\frac{1}{2}}}{2}\right]^2, & 0 < \omega \le \omega^*_{\xi,\eta}, \end{cases}$$

where

$$\omega^*_{\xi,\eta} = \frac{2}{1+(1-\alpha^2_{\xi,\eta})^{\frac{1}{2}}}.$$

It is easy to see from (3.4) that when $0 < \omega_{\xi,\eta} < 2$, $\rho_{\xi,\eta} < 1$. In addition, the relaxation parameter $\omega = \omega^*_{\xi,\eta}$ minimizes $\rho_{\xi,\eta}$, which takes the value $\omega^*_{\xi,\eta} - 1$.

Next, let us vary the values of ξ and η, and determine the optimal relaxation parameter for $(\xi, \eta) \in K$. Since the procedure is standard, only the results are summarized [23],[31]. The optimal relaxation parameter is

$$(3.5)\qquad \omega^* = \frac{2}{1+(1-\alpha^2_{\xi,\eta,\max})^{\frac{1}{2}}}, \quad \alpha_{\xi,\eta,\max} = \max_{(\xi,\eta)\in K} \alpha_{\xi,\eta} = \cos(\pi h),$$

where $\alpha_{\xi,\eta,\max}$ occurs at the lowest frequency $(\xi, \eta) = (1,1)$. Its corresponding spectral radius is

$$(3.6)\qquad \rho^*_{\text{SOR}}(\text{red/black ordering ; Dirichlet b.c.}) = \omega^* - 1 \approx 1 - 2\pi h.$$

With this optimal relaxation parameter ω^*, the eigenvalues of S_{rb} are distributed along a circle of radius $\omega^* - 1$ in the complex plane. The results in (3.5) and (3.6) are in fact special cases of the general SOR theory by Young [30],[31].

3.2. SSOR iteration. One SSOR iteration with the red/black ordering consists of one red/black SOR iteration followed by one black/red SOR iteration. Hence, the corresponding iteration matrix can be written as

$$(3.7)\qquad \hat{S}_{\text{SSOR}}(\xi,\eta,\omega) = \hat{S}_r(\xi,\eta,\omega)\hat{S}_b(\xi,\eta,\omega)\hat{S}_b(\xi,\eta,\omega)\hat{S}_r(\xi,\eta,\omega),$$

where $\hat{S}_r$ and $\hat{S}_b$ are given in (3.2). Note that we can rewrite the frequency domain red/black SOR iteration matrix as

$$(3.8)\qquad \hat{S}_b(\xi,\eta,\omega)\hat{S}_r(\xi,\eta,\omega) = I - \omega(I - \omega\hat{L}(\xi,\eta))^{-1}\hat{A}(\xi,\eta),$$

where I is the 2-by-2 identity matrix, $\hat{A}(\xi,\eta)$ is the frequency domain Laplacian defined by (2.10), and

$$\hat{L}(\xi,\eta) = \begin{bmatrix} 0 & 0 \\ \alpha_{\xi,\eta} & 0 \end{bmatrix}.$$

Similarly, the frequency domain black/red SOR iteration matrix can be written as

$$(3.9)\qquad \hat{S}_r(\xi,\eta,\omega)\hat{S}_b(\xi,\eta,\omega) = I - \omega(I - \omega\hat{U}(\xi,\eta))^{-1}\hat{A}(\xi,\eta),$$

where $\hat{U}(\xi,\eta) = \hat{L}^T(\xi,\eta)$. Combining (3.7)-(3.9), we have

$$(3.10)\quad \hat{S}_{\text{SSOR}}(\xi,\eta,\omega) = I - \omega(2-\omega)(I - \omega\hat{U}(\xi,\eta))^{-1}(I - \omega\hat{L}(\xi,\eta))^{-1}\hat{A}(\xi,\eta).$$

The optimal relaxation parameter is selected to minimize the spectral radius of S_{SSOR}, or equivalently, to maximize the smaller eigenvalue of the second term in the right-hand side of (3.10). It is easy to see that $\omega(2-\omega)$ takes the maximum value when $\omega = 1$. In addition, it will be shown in §4.1 that $\omega = 1$ maximizes the smaller eigenvalue $\lambda_{\xi,\eta,-}$ of the matrix

$$(I - \omega\hat{U}(\xi,\eta))^{-1}(I - \omega\hat{L}(\xi,\eta))^{-1}\hat{A}(\xi,\eta),$$

for $(\xi,\eta) \in K$. Thus, the optimal relaxation parameter is 1, with which the spectral radius of the SSOR iteration becomes

$$\rho^*_{\text{SSOR}}(\text{red/black ordering ; Dirichlet b.c.}) = \cos^2 \pi h \approx 1 - \pi^2 h^2. \tag{3.11}$$

4. Analysis of preconditioners.

4. Analysis of preconditioners. An important class of iterative methods for solving elliptic PDEs is obtained by first preconditioning the system of equations and then solving the preconditioned system with effective iterative methods [9]. One such example is the preconditioned conjugate gradient (PCG) method. It is well known that the rate of convergence of a preconditioned iterative method depends on the condition number as well as the distribution of the eigenvalues of the preconditioned system [7],[9].

For the model Poisson problem with the natural ordering, Chan and Elman [13] used Fourier analysis with basis functions $e^{i2\pi\xi jh}e^{i2\pi\eta kh}$ to obtain all eigenvalues of the preconditioned Laplacian with the ILU, MILU, and SSOR preconditioners. Here, we analyze the eigenstructure of the model problem (2.5) with the red/black ordering. The two-color Fourier analysis with basis functions $\sin(\xi\pi jh)\sin(\eta\pi kh)$ is used to determine all eigenvalues of the preconditioned system. Note that different basis functions are chosen for these two orderings. For the red/black ordering, since the stencils of iterative operators are symmetric, either sine or complex sinusoidal functions can be conveniently used as basis functions, and the resulting analysis is exact for Dirichlet and periodic boundary conditions, respectively. For the natural ordering, since the stencils of iterative operators are usually not symmetric, only the complex sinusoidal functions can be conveniently used as basis functions. Such an analysis is exact for periodic boundary conditions but, in general, is not exact for Dirichlet boundary conditions. However, experimental results indicate that the eigenvalue distribution of the preconditioned system is not sensitive to the change of boundary conditions [13].

Three different types of preconditioners, i.e., the SSOR, ILU, and MILU preconditioners, are studied below.

4.1. SSOR preconditioner.

4.1. SSOR preconditioner. Suppose that we define the following local operators:

$$L_{j,k} = \begin{cases} 0, & (j,k) \text{ red}, \\ \frac{1}{4}(E_x + E_x^{-1} + E_y + E_y^{-1}), & (j,k) \text{ black}, \end{cases} \tag{4.1a}$$

$$U_{j,k} = \begin{cases} \frac{1}{4}(E_x + E_x^{-1} + E_y + E_y^{-1}), & (j,k) \text{ red}, \\ 0, & (j,k) \text{ black}. \end{cases} \tag{4.1b}$$

It is easy to see that their corresponding global operators L and U are related to A by $A = I - (L+U)$. Then, the global SSOR preconditioner with the red/black ordering is of the form [6]

$$Q_S = (I - \omega L)(I - \omega U), \tag{4.2}$$

where ω is the relaxation parameter. By using the two-color Fourier analysis, we can transform it to the frequency domain

$$(4.3)\quad \hat{Q}_S(\xi,\eta) = \begin{bmatrix} 1 & 0 \\ -\omega\alpha_{\xi,\eta} & 1 \end{bmatrix} \begin{bmatrix} 1 & -\omega\alpha_{\xi,\eta} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -\omega\alpha_{\xi,\eta} \\ -\omega\alpha_{\xi,\eta} & 1+\omega^2\alpha_{\xi,\eta}^2 \end{bmatrix},$$

where $\alpha_{\xi,\eta}$ is defined as in (2.6b). From (2.10) and (4.3), we find that the SSOR preconditioned operator $Q_S^{-1}A$ has the spectral representation

$$(4.4)\quad \hat{Q}_S^{-1}(\xi,\eta)\hat{A}(\xi,\eta) = \begin{bmatrix} 1-\omega\alpha_{\xi,\eta}^2+\omega^2\alpha_{\xi,\eta}^2 & -\alpha_{\xi,\eta}+\omega\alpha_{\xi,\eta}-\omega^2\alpha_{\xi,\eta}^3 \\ -\alpha_{\xi,\eta}+\omega\alpha_{\xi,\eta} & 1-\omega\alpha_{\xi,\eta}^2 \end{bmatrix},$$

which has two eigenvalues

$$(4.5)\quad \lambda_{\xi,\eta,\pm} = 1-\frac{1}{2}\alpha_{\xi,\eta}^2\omega(2-\omega) \pm \frac{1}{2}\alpha_{\xi,\eta}[\alpha_{\xi,\eta}^2\omega^2(2-\omega)^2-4\omega(2-\omega)+4]^{\frac{1}{2}}.$$

When $0<\omega<2$, the eigenvalues $\lambda_{\xi,\eta}$ are not only real but also positive and, therefore, $Q_S^{-1}A$ corresponds to a symmetric positive definite (SPD) matrix suitable for the conjugate gradient method. The condition number κ of the operator $Q_S^{-1}A$ is determined by

$$\kappa(Q_S^{-1}A) = \frac{\max_{\xi,\eta}|\lambda_{\xi,\eta,+}|}{\min_{\xi,\eta}|\lambda_{\xi,\eta,-}|},$$

which is to be minimized by choosing an appropriate relaxation parameter $0<\omega<2$.

To determine the condition number, it is convenient to rewrite (4.5) as

$$\lambda_\pm(x,y) = 1-\frac{x^2y}{2} \pm \frac{x}{2}(x^2y^2-4y+4)^{\frac{1}{2}},$$

where $0\le x=\alpha_{\xi,\eta}<1$ and $0<y=\omega(2-\omega)\le 1$. By taking the partial derivative with respect to y for $\lambda_\pm$, we find that λ_+ and λ_- are monotonically decreasing and increasing, respectively, for given x. So, $y=1$ gives the smallest condition number and the optimal relaxation parameter is 1. The corresponding eigenvalues in (4.5) become

$$(4.6)\qquad \lambda_{\xi,\eta,\pm} = 1-\frac{1}{2}\alpha_{\xi,\eta}^2 \pm \frac{1}{2}\alpha_{\xi,\eta}^2.$$

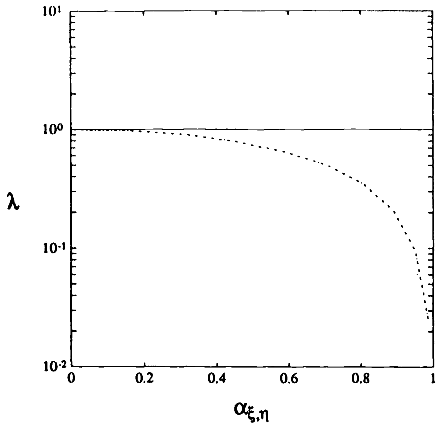
The maxima of $\lambda_{\xi,\eta,+}$ are 1, and the minimum of $\lambda_{\xi,\eta,-}$ is $1-\cos^2(\pi h)\approx\pi^2h^2$, which occurs at $(\xi,\eta)=(1,1)$. Therefore, the condition number of the SSOR preconditioned Laplacian is

$$(4.7)\qquad \kappa(Q_S^{-1}A) = \frac{1}{1-\cos^2(\pi h)} \approx \frac{1}{\pi^2h^2} = O(h^{-2}).$$

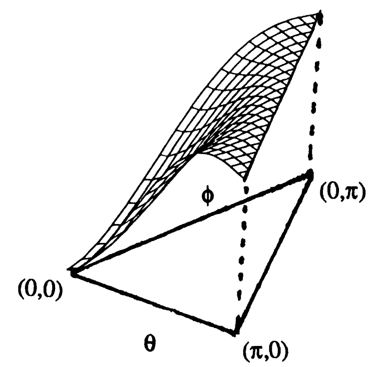
The distribution of the eigenvalues $\lambda_{\xi,\eta,\pm}$ given by (4.6) is plotted as a function of $\alpha_{\xi,\eta}$ in Fig. 2(a). The surface plot of the eigenvalue $\lambda_{\xi,\eta,-}$ as a function of $(\theta,\phi)=(\xi\pi h,\eta\pi h)$ is presented in Fig. 2(b). Note that the condition number of the Laplacian is

$$\frac{1+\cos(\pi h)}{1-\cos(\pi h)} \approx \frac{4}{\pi^2h^2}.$$

Hence, for small h, the red/black SSOR preconditioner only reduces the condition number of the original matrix by a factor of 4.



(a)



(b)

FIG. 2. (a) *The eigenvalues* $\lambda_\pm$ *of the* SSOR-*preconditioned system as functions of* $\alpha_{\xi,\eta}$ *and* (b) *the surface plot of* λ_+ *as a function of* (θ, ϕ) *with* $h = 0.05$.

4.2. ILU preconditioner. An incomplete factorization for a matrix can be determined by imposing specific sparse patterns and constraints for elements on the factorizing matrices as well as their product. Since the construction of a matrix specifies not only a system of equations but also an ordering scheme for the variables, the incomplete factorization depends highly on the ordering. In this and the following sections, we study the spectra of two well-known preconditioners, i.e., the ILU and MILU preconditioners, which are constructed based on incomplete lower/upper triangular factorization.

The ILU and MILU factorizations were originally defined in [25] and [17], respectively. We summarize their definitions as follows. It is required for both the ILU and MILU factorizations that the factorizing lower and upper triangular matrices have the same sparse patterns as the lower and upper triangular parts of the original matrix. Besides, the off-diagonal nonzero elements of the original matrix should have the same values as the corresponding elements of the product matrix. The major difference between them is that the ILU factorization requires that the diagonal elements of the original and product matrices be also the same whereas the MILU factorization requires that the row sum of the product matrix differ from the row sum of the original matrix by a small quantity $\delta = ch^2$, where c is a constant independent of h.

The factorizing operators generally have different coefficients associated with different grid points due to the boundary effects. However, these coefficients usually reach their asymptotic constant values for the region sufficiently far away from boundaries. In the following analysis, we ignore the boundary effect and analyze the preconditioned system with the asymptotic preconditioners.

For the ILU factorization, consider the following local operators:

$$L_{j,k} = \begin{cases} 1, & (j,k) \text{ red}, \\ 1 - \frac{1}{4}(E_x + E_x^{-1} + E_y + E_y^{-1}), & (j,k) \text{ black}, \end{cases}$$

$$U_{j,k} = \begin{cases} 1 - \frac{1}{4}(E_x + E_x^{-1} + E_y + E_y^{-1}), & (j,k) \text{ red}, \\ \frac{3}{4}, & (j,k) \text{ black}. \end{cases}$$

With the red/black ordering, the global operators L and U correspond to lower and upper triangular matrices. Since the operator $L_{j,k}$ (or $U_{j,k}$) has nonzero coefficients for the terms 1, E_x, E_x^{-1}, E_y, and E_y^{-1}, the sparse pattern of L (or U) is the same as that of the original matrix A for the lower (or upper) triangular part. We define the global operator Q_I to be the product of the lower and upper global operators

$$Q_I = LU.$$

Let $R = Q_I - A$. Then R consists of the local operators

$$R_{j,k} = \begin{cases} 0, & (j,k) \text{ red}, \\ \frac{1}{8}(E_xE_y + E_x^{-1}E_y + E_xE_y^{-1} + E_x^{-1}E_y^{-1}) & \\ \quad + \frac{1}{16}(E_x^2 + E_x^{-2} + E_y^2 + E_y^{-2}), & (j,k) \text{ black}, \end{cases}$$

for points not close to the boundaries. Note that the operator $(Q_I)_{j,k}$ has the same coefficients as $A_{j,k}$ (i.e., $R_{j,k} = 0$) for terms corresponding to 1, E_x, E_x^{-1}, E_y, and E_y^{-1}, which constitute the nonzero terms for the sparse matrix A. Note that the sparse patterns of L, U, and Q_I described above are consistent with the sparsity conditions required by the ILU factorization. We conclude that Q_I is the desired ILU preconditioner for the Laplacian with the red/black ordering.

In the Fourier domain, we have

$$\hat{Q}_I(\xi,\eta)=\begin{bmatrix} 1 & 0 \\ -\alpha_{\xi,\eta} & 1 \end{bmatrix}\begin{bmatrix} 1 & -\alpha_{\xi,\eta} \\ 0 & \frac{3}{4} \end{bmatrix}=\begin{bmatrix} 1 & -\alpha_{\xi,\eta} \\ -\alpha_{\xi,\eta} & \frac{3}{4}+\alpha_{\xi,\eta}^2 \end{bmatrix}.$$

Therefore, the ILU preconditioned operator $Q_I^{-1}A$ has the spectral representation

$$\hat{Q}_I^{-1}(\xi,\eta)\hat{A}(\xi,\eta)=\begin{bmatrix} 1 & \frac{1}{3}\alpha_{\xi,\eta}-\frac{4}{3}\alpha_{\xi,\eta}^3 \\ 0 & \frac{4}{3}(1-\alpha_{\xi,\eta}^2) \end{bmatrix},$$

which has two real and positive eigenvalues

$$\lambda_{\xi,\eta}=1,\ \frac{4}{3}(1-\alpha_{\xi,\eta}^2). \tag{4.8}$$

The condition number of the ILU preconditioned Laplacian can be determined by

$$\kappa(Q_I^{-1}A)=\frac{\max_{\xi,\eta}|\lambda_{\xi,\eta}|}{\min_{\xi,\eta}|\lambda_{\xi,\eta}|}=\frac{\max_{\xi,\eta}\frac{4}{3}(1-\alpha_{\xi,\eta}^2)}{\min_{\xi,\eta}\frac{4}{3}(1-\alpha_{\xi,\eta}^2)}\approx\frac{1}{\pi^2h^2}=O(h^{-2}), \tag{4.9}$$

where the maximum value $\frac{4}{3}$ occurs when $\alpha_{\xi,\eta}=0$ and the minimum value $\frac{4}{3}[1-\cos^2(\pi h)]$ occurs at $(\xi,\eta)=(1,1)$. By the ILU preconditioning, we reduce the condition number of A approximately by a factor of 4. The distribution of the two eigenvalues $\lambda_{\xi,\eta}$ (4.8) as a function of $\alpha_{\xi,\eta}$ and the surface plot of the eigenvalue $\frac{4}{3}(1-\alpha_{\xi,\eta}^2)$ as a function of $(\theta,\phi)=(\xi\pi h,\eta\pi h)$ are shown in Figs. 3(a) and 3(b). The corresponding plot of the natural ordering case can be found in [13], where the condition number of A is reduced approximately by a factor $2(2+\sqrt{2})$.

4.3. MILU preconditioner. For the MILU factorization, consider the following local operators:

$$L_{j,k}=\begin{cases} 1+\delta, & (j,k)\text{ red},\\ 1+\delta-\frac{1}{1+\delta}-\frac{1}{4}(E_x+E_x^{-1}+E_y+E_y^{-1}), & (j,k)\text{ black},\end{cases}$$
$$U_{j,k}=\begin{cases} 1-\frac{1}{4(1+\delta)}(E_x+E_x^{-1}+E_y+E_y^{-1}), & (j,k)\text{ red},\\ 1, & (j,k)\text{ black},\end{cases}$$

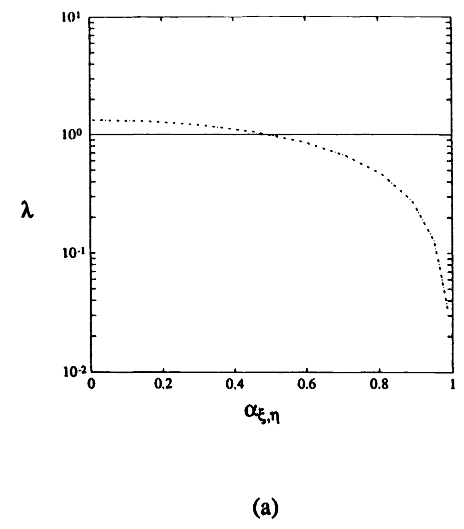
where $\delta=ch^2$. The sparse patterns of L and U given above are the same as those for the ILU factorization, but they have different weighting coefficients. The global operator Q_M is defined to be the product of the lower and upper global operators

$$Q_M=LU.$$

Let $R=Q_M-A$. Then R consists of the local operators

$$R_{j,k}=\begin{cases} \delta, & (j,k)\text{ red},\\ \delta+\frac{1}{1+\delta}\left[-\frac{3}{4}+\frac{1}{8}(E_xE_y+E_x^{-1}E_y+E_xE_y^{-1}+E_x^{-1}E_y^{-1})\right. & \\ \left.\quad+\frac{1}{16}(E_x^2+E_x^{-2}+E_y^2+E_y^{-2})\right], & (j,k)\text{ black},\end{cases}$$

for points not close to the boundaries. Note that $(Q_M)_{j,k}$ has the same coefficients as $A_{j,k}$ ($R_{j,k}=0$) for terms E_x, E_x^{-1}, E_y and E_y^{-1}, which are nonzero off-diagonal entries of the matrix A. However, unlike the ILU case, the matrices A and Q_M do not have the same diagonal entries. Instead, we find that the sum of coefficients of



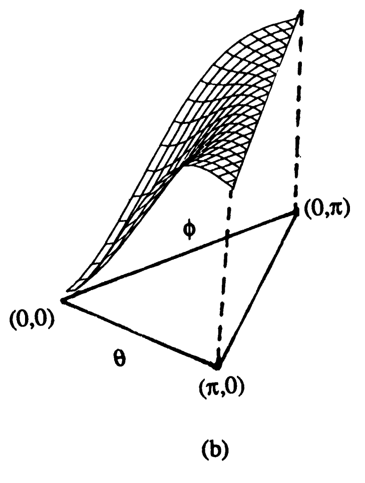


FIG. 3. (a) *The eigenvalues of the* ILU*-preconditioned system as functions of* $\alpha_{\xi,\eta}$ *and* (b) *the surface plot of the nonconstant eigenvalue as a function of* (θ, ϕ) *with* $h = 0.05$.

the local error operator $R_{j,k}$ equals δ. This implies that the row sums of matrices A and Q_M differ by a quantity of δ. By definition, Q_M is the MILU preconditioner for the Laplacian with the red/black ordering.

The Fourier transform of Q_M gives

$$\begin{aligned}\hat{Q}_M(\xi,\eta) &= \begin{bmatrix} 1+\delta & 0 \\ -\alpha_{\xi,\eta} & 1+\delta-\frac{1}{1+\delta} \end{bmatrix}\begin{bmatrix} 1 & -\frac{\alpha_{\xi,\eta}}{1+\delta} \\ 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1+\delta & -\alpha_{\xi,\eta} \\ -\alpha_{\xi,\eta} & 1+\delta+\frac{\alpha^2_{\xi,\eta}-1}{1+\delta} \end{bmatrix}.\end{aligned}$$

Hence, the MILU preconditioned operator has the spectral representation

$$\hat{Q}_M^{-1}(\xi,\eta)\hat{A}(\xi,\eta) = \frac{1}{\delta(\delta+2)}\begin{bmatrix} \delta(1+\frac{1-\alpha^2_{\xi,\eta}}{1+\delta}) & -\alpha_{\xi,\eta}(\delta+\frac{\alpha^2_{\xi,\eta}-1}{1+\delta}) \\ -\alpha_{\xi,\eta}\delta & 1+\delta-\alpha^2_{\xi,\eta} \end{bmatrix},$$

which has the eigenvalues

(4.10)
$$\lambda_{\xi,\eta,\pm} = \frac{2\delta(1+\delta)+(1-\alpha^2_{\xi,\eta})(1+2\delta)\pm[(1-\alpha^2_{\xi,\eta}-2\delta\alpha^2_{\xi,\eta})^2+4\delta^3\alpha^2_{\xi,\eta}(2+\delta)]^{\frac{1}{2}}}{2\delta(1+\delta)(2+\delta)}.$$

Note that if $\delta = 0$ (i.e., $c = 0$), $\hat{Q}_M(\xi,\eta)$ is a singular matrix that cannot be used as a preconditioner. For $c > 0$, since

$$4\delta^3\alpha^2_{\xi,\eta}(2+\delta) \ll (1-\alpha^2_{\xi,\eta}-2\delta\alpha^2_{\xi,\eta})^2, \tag{4.11}$$

as h goes to zero, we can simplify (4.10) as follows:

$$\lambda_{\xi,\eta,\pm} \approx \frac{2\delta(1+\delta)+(1-\alpha^2_{\xi,\eta})(1+2\delta)\pm(1-\alpha^2_{\xi,\eta}-2\delta\alpha^2_{\xi,\eta})}{2\delta(1+\delta)(2+\delta)}.$$

For small h and positive c, $1-\alpha^2_{\xi,\eta}-2\delta\alpha^2_{\xi,\eta}$ is positive. So, $\lambda_{\xi,\eta,+}$ and $\lambda_{\xi,\eta,-}$ are the larger and smaller eigenvalues, respectively. Then, the condition number of the MILU preconditioned Laplacian is found to be

$$\begin{aligned}\kappa(Q_M^{-1}A) &= \frac{\max_{\xi,\eta}|\lambda_{\xi,\eta,+}|}{\min_{\xi,\eta}|\lambda_{\xi,\eta,-}|} \\ &\approx \frac{\max_{\xi,\eta}\delta(1-\alpha^2_{\xi,\eta}+\delta)+(1-\alpha^2_{\xi,\eta})(1+\delta)}{\delta(2+\delta)} \approx \frac{1}{2ch^2} = O(h^{-2}),\end{aligned} \tag{4.12}$$

where the maximum value $(1+\delta)^2$ occurs when $\alpha_{\xi,\eta} = 0$.

For fixed h, the optimal parameter c and the corresponding condition number $\kappa(Q_M^{-1}A)$ can be determined by solving (4.10) numerically. The condition number $\kappa(Q_M^{-1}A)$ is plotted as a function of the parameter c with different h in Fig. 4.

For small c ($c \leq 5$), the condition number behaves very close to $(2ch^2)^{-1}$ as predicted by (4.12). For $c \gg 5$, condition (4.11) is no more valid, and we see that $\kappa(Q_M^{-1}A)$ remains approximately the same for a wide range of c. Thus, the condition number is not sensitive to the selection of the relaxation parameter, as long as it is in the appropriate range. For these values of h used in Fig. 4, the optimal condition number is achieved when $c \approx 5$. Thus, we know from the above analysis that the condition number of the original Laplacian is improved approximately by a factor of

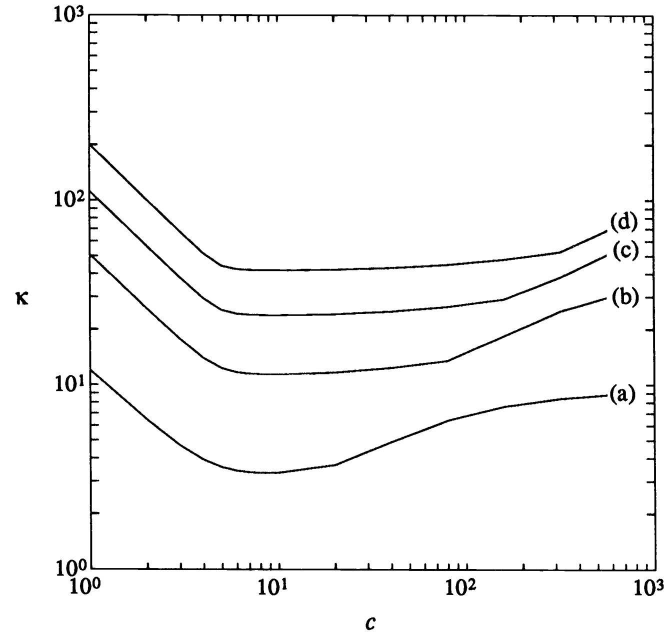


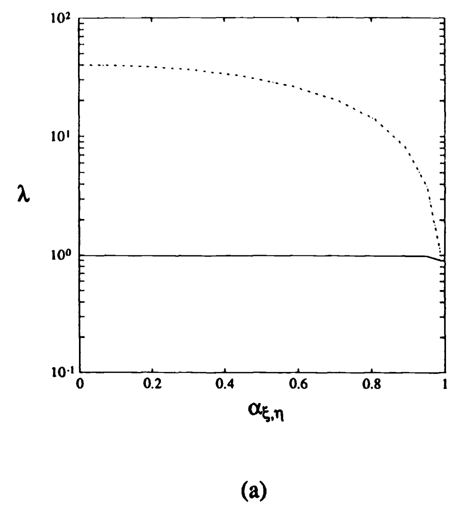
FIG. 4. *The condition number of the* MILU*-preconditioned system as a function of the parameter with* (a) $h = 1/5$, (b) $h = 1/10$, (c) $h = 1/15$, *and* (d) $h = 1/20$.

$8c/\pi^2 \approx 4$. This improvement is about the same as that for the red/black ordered SSOR and ILU preconditioners.

The distribution of the eigenvalues $\lambda_{\xi,\eta,\pm}$ given by (4.10) with $\delta = 5h^2$ (i.e., $c = 5$) is plotted as a function of $\alpha_{\xi,\eta}$ in Fig. 5(a). Note that the eigenvalue $\lambda_{\xi,\eta,-}$ is nearly a constant. The surface plot of the eigenvalue $\lambda_{\xi,\eta,+}$ as a function of $(\theta, \phi) = (\xi\pi h, \eta\pi h)$ is shown in Fig. 5(b).

5. Analysis of the multigrid (MG) method. Multigrid (MG) methods provide one of the most effective ways for solving elliptic PDEs. The multigrid iteration is often modeled by a $(h, 2h)$ two-grid iteration process so that its mechanism can be easily understood. The efficiency of the two-grid (or multigrid) iteration is based on a simple idea – to treat error components of low and high frequencies differently. Suppose that we partition the Fourier domain into two regions of which the low frequency region contains $1 \leq \xi, \eta < M/2$ and the high frequency region contains $M/2 \leq \xi \leq M - 1$ or $M/2 \leq \eta \leq M - 1$. The mechanism of the two-grid iteration with the damped Jacobi or the lexicographical (naturally ordered) Gauss-Seidel smoother can be easily explained. That is, the high frequency error is smoothed at the fine grid, whereas the low frequency error is corrected at the coarse grid. Thus, the study of the error smoothing over the high frequency region provides a rough estimate of the convergence behavior of the multigrid iteration. This is known as the *smoothing rate analysis* [10].

It is known that MG with the red/black Gauss-Seidel smoother performs better than MG with the damped Jacobi or the lexicographical Gauss-Seidel smoother for the model Poisson problem [29]. However, the efficiency of the red/black Gauss-Seidel smoother cannot be appropriately explained by the smoothing rate analysis.



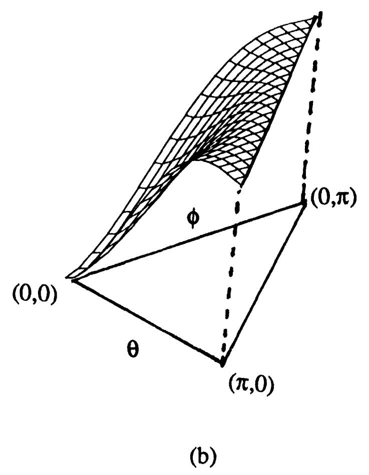


FIG. 5. (a) *The eigenvalues of the* MILU*-preconditioned system as functions of* $\alpha_{\xi,\eta}$ *and* (b) *the surface plot of the nonconstant eigenvalue as a function of* (θ, ϕ) *with* $h = 0.05$.

To see this, let us examine the red/black Gauss-Seidel iteration matrix in the two-color Fourier domain

$$\hat{S}_{\mathrm{RBGS}}(\xi,\eta)=\begin{bmatrix} 0 & \alpha_{\xi,\eta} \\ 0 & \alpha_{\xi,\eta}^2 \end{bmatrix},$$

which is obtained from (3.3) with $\omega = 1$. The smoothing rate μ is usually defined as

$$\mu = \max_{(\xi,\eta)\in K_{\mathrm{high}}} \rho[\hat{S}_{\mathrm{RBGS}}(\xi,\eta)] = \cos^2(\pi h) \approx 1-\pi^2h^2,$$

where

$$K_{\mathrm{high}} = \left\{(\xi,\eta) : \xi,\eta \in I, \frac{M}{2} \le \xi \le M-1 \text{ or } \frac{M}{2} \le \eta \le M-1\right\},$$

and the maximum value occurs at $\xi = \eta = M-1$. This shows that the red/black Gauss-Seidel smoother has a very poor smoothing rate as compared to the natural ordering case for which the smoothing rate is $\frac{1}{2}$ [10].

Since the smoothing rate analysis does not explain how the MG method with the red/black Gauss-Seidel smoothing works, it is essential to perform a complete two-grid analysis, which includes both smoothing and coarse-grid correction. A two-grid analysis was performed by Stüben and Trottenberg by using modified Fourier analysis [29]. Here, we use the framework of two-color Fourier analysis to analyze this method. Our objective is to give a clearer explanation of the physical mechanism behind this method rather than to rederive the specific result obtained in [29]. We will show that the two-color two-grid iteration process asymptotically reduces to a one-color two-grid iteration process that is much easier to understand.

5.1. Framework of the two-color two-grid analysis. We summarize the two-grid iteration model, which is discussed in detail in [29], as follows. Let L_h and L_{2h} be the 5-point discretizations of the Laplacian on grids Ω_h and Ω_{2h}. Consider the full-weighting restriction operator I_h^{2h} from Ω_h to Ω_{2h} and the linear interpolation operator I_{2h}^h from Ω_{2h} to Ω_h, which are usually represented in stencil form as

$$(5.1)\qquad I_h^{2h} : \left|\begin{matrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{matrix}\right|_h^{2h}, \qquad I_{2h}^h : \left|\begin{matrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{matrix}\right|_{2h}^h .$$

Then, a $(h,2h)$ two-grid iteration matrix with the red/black Gauss-Seidel smoothing can be written as

$$(5.2)\qquad T_h^{2h} = (S_{\mathrm{RBGS}})^{\nu_2} K_h^{2h} (S_{\mathrm{RBGS}})^{\nu_1}, \quad K_h^{2h} = I_h - I_{2h}^h L_{2h}^{-1} I_h^{2h} L_h,$$

where I_h is the identity matrix, ν_1 and ν_2 are the numbers of presmoothing and postsmoothing iterations. We want to determine the spectral radius $\rho(T_h^{2h})$ and, more importantly, to explain how the two-grid iteration (5.2) works.

In the current context, (ξ,η) is nondegenerate if $1 \le \xi,\eta < M/2$ and degenerate if $\xi = M/2$ or $\eta = M/2$. We consider only the nondegenerate case, and the degenerate case can be treated similarly [21]. Let $\hat{e}_{\xi,\eta}$ be the Fourier coefficient of the error, and let $\hat{r}_{\xi,\eta}$ and $\hat{b}_{\xi,\eta}$ be the Fourier coefficients of the error defined at the red and black points, respectively. Through the iteration (5.2), four Fourier components $\hat{e}_{\xi,\eta}$,

$\hat{e}_{\xi,M-\eta}$, $\hat{e}_{M-\xi,\eta}$, and $\hat{e}_{M-\xi,M-\eta}$ with $1 \le \xi, \eta < M/2$, are coupled together. Hence, the spectrum of T_h^{2h} can be analyzed by focusing on a subspace spanned by these four components. Stüben and Trottenberg used the unit vector of these four Fourier components as a basis. Here, we use a different basis obtained by

$$\begin{pmatrix} \hat{r}_{\xi,\eta} \\ -\hat{r}_{\xi',\eta'} \\ \hat{b}_{\xi,\eta} \\ -\hat{b}_{\xi',\eta'} \end{pmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & -1 & -1 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix} \begin{pmatrix} \hat{e}_{\xi,\eta} \\ \hat{e}_{M-\xi,M-\eta} \\ \hat{e}_{\xi',\eta'} \\ \hat{e}_{M-\xi',M-\eta'} \end{pmatrix},$$

where

$$(\xi', \eta') = \begin{cases} (M-\xi, \eta) & \text{if } \xi \ge \eta, \\ (\xi, M-\eta) & \text{if } \xi < \eta. \end{cases}$$

Note that the new basis is basically obtained by folding the conventional Fourier domain into two-color Fourier domain as shown in (2.4), and therefore the above transformation maps the coupled four Fourier components $\hat{e}_{\xi,\eta}$, $\hat{e}_{\xi,M-\eta}$, $\hat{e}_{M-\xi,\eta}$, and $\hat{e}_{M-\xi,M-\eta}$ into red and black waves with indices (ξ,η) and (ξ',η') (Fig. 6).

We choose the convention that each 4×4 frequency domain matrix describes a mapping from a vector space spanned by

$$(\hat{r}_{\xi,\eta}, -\hat{r}_{\xi',\eta'}, \hat{b}_{\xi,\eta}, -\hat{b}_{\xi',\eta'})^T$$

onto itself for the rest of this section. To simplify the notation, the abbreviations

$$\alpha = \tfrac{1}{2}(\cos\xi\pi h + \cos\eta\pi h), \quad \tilde{\alpha} = \tfrac{1}{2}(\cos\xi'\pi h + \cos\eta'\pi h),$$
$$\beta = \cos\xi\pi h \cos\eta\pi h, \quad \tilde{\beta} = \cos\xi'\pi h \cos\eta'\pi h,$$

are used. We also omit the subscripts ξ, η, ξ' and η' for α, $\tilde{\alpha}$, β and $\tilde{\beta}$ and the arguments ξ, η for frequency domain matrices.

5.2. Analysis of elements for two-grid iteration. The building blocks for the two-grid iteration process (5.2) are analyzed in this section. In the two-color Fourier domain, the red/black Gauss-Seidel iteration can be represented by

$$\hat{S}_{\mathrm{RBGS}} = \begin{bmatrix} I & 0 \\ \hat{J} & 0 \end{bmatrix} \begin{bmatrix} 0 & \hat{J} \\ 0 & I \end{bmatrix} = \begin{bmatrix} 0 & \hat{J} \\ 0 & \hat{J}^2 \end{bmatrix}, \tag{5.3}$$

where 0 is the 2×2 zero matrix, I is the 2×2 identity matrix, and

$$\hat{J} = \begin{bmatrix} \alpha & 0 \\ 0 & \tilde{\alpha} \end{bmatrix}.$$

In addition, the frequency domain matrices for operators I_h, L_h, and L_{2h}^{-1} in (5.2) are

$$\hat{I}_h = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}, \quad \hat{L}_h = \frac{4}{h^2} \begin{bmatrix} -I & \hat{J} \\ \hat{J} & -I \end{bmatrix}, \tag{5.4a}$$

and

$$\hat{L}_{2h}^{-1} = \frac{h^2}{2\delta}, \quad \delta = 2\alpha^2 - \beta - 1. \tag{5.4b}$$

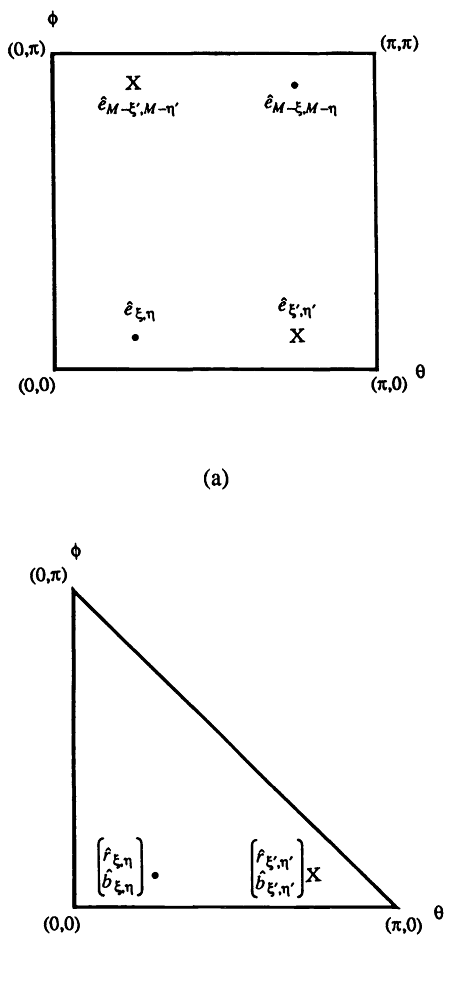

ϕ
$(0,\pi)$
(π,π)
X
$\hat{e}_{M-\xi',M-\eta'}$
$\hat{e}_{M-\xi,M-\eta}$
$\hat{e}_{\xi,\eta}$
$\hat{e}_{\xi',\eta'}$
X
$(0,0)$
$(\pi,0)$
θ
(a)
ϕ
$(0,\pi)$
$\begin{bmatrix}\hat{r}_{\xi,\eta}\\ \hat{b}_{\xi,\eta}\end{bmatrix}$
$\begin{bmatrix}\hat{r}_{\xi',\eta'}\\ \hat{b}_{\xi',\eta'}\end{bmatrix}$ X
$(0,0)$
$(\pi,0)$
θ
(b)

FIG. 6. *Four coupled Fourier components in* (a) *conventional and* (b) *two-color Fourier domains.*

In (5.3) and (5.4), there is no coupling between vectors $(\hat{r}_{\xi,\eta}, \hat{b}_{\xi,\eta})^T$ and $(\hat{r}_{\xi',\eta'}, \hat{b}_{\xi',\eta'})^T$. The coupling between them comes from the full-weighting restriction and linear interpolation operations. The decomposition, commonly used in the multirate signal processing context [15], is very useful for understanding the physical mechanism of interpolation and restriction operators, and for deriving their frequency domain matrices. Conceptually, we decompose the restriction procedure I_{2h}^{h} into two steps.

Step 1. Lowpass filtering (or averaging) at every point of Ω_h, where the weighting coefficients are specified by stencil I_h^{2h} of (5.1).

Step 2. Down-sampling (or injecting) values from Ω_h to Ω_{2h}.

The interpolation operator I_{2h}^{h} is also decomposed into two steps.

Step 1. Up-sampling values from Ω_{2h} to Ω_h, by which we assign 0 to points that belong to $\Omega_h - \Omega_{2h}$

Step 2. Lowpass filtering at every point of Ω_h, where the weighting coefficients are specified by stencil I_{2h}^{h} of (5.1).

It is relatively easy to find a frequency domain matrix representation for each of the above steps. Combining them together, we obtain

$$\hat{I}_h^{2h} = [1\ 1\ 0\ 0] \times \frac{1}{4} \begin{bmatrix} 1+\beta & 0 & 2\alpha & 0 \\ 0 & 1+\tilde{\beta} & 0 & 2\tilde{\alpha} \\ 2\alpha & 0 & 1+\beta & 0 \\ 0 & 2\tilde{\alpha} & 0 & 1+\tilde{\beta} \end{bmatrix} = \frac{1}{4}[1+\beta\ 1+\tilde{\beta}\ 2\alpha\ 2\tilde{\alpha}],$$

and

$$\hat{I}_{2h}^{h} = \begin{bmatrix} 1+\beta & 0 & 2\alpha & 0 \\ 0 & 1+\tilde{\beta} & 0 & 2\tilde{\alpha} \\ 2\alpha & 0 & 1+\beta & 0 \\ 0 & 2\tilde{\alpha} & 0 & 1+\tilde{\beta} \end{bmatrix} \times \frac{1}{2} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1+\beta \\ 1+\tilde{\beta} \\ 2\alpha \\ 2\tilde{\alpha} \end{bmatrix}.$$

Note that in the frequency domain the down-sampling operation adds the high frequency component $-\hat{r}_{\xi',\eta'}$ to the low frequency component $\hat{r}_{\xi,\eta}$. This phenomenon is known as *aliasing* [15]. On the other hand, the up-sampling operation duplicates the low frequency component $\hat{r}_{\xi,\eta}$ in the high frequency region in the form of $-\hat{r}_{\xi',\eta'}$, which is called *imaging* [15]. The lowpass filters cascaded with the down-sampling and the up-sampling operators are basically used to reduce the aliasing and imaging effects. For example, for low frequency components with $\xi\pi h$ and $\eta\pi h$ close to 0, we have $\alpha \approx 1$, $\beta \approx 1$, $\tilde{\alpha} \approx 0$, and $\tilde{\beta} \approx -1$. Hence, the aliasing and imaging effects occurring between $(\hat{r}_{\xi,\eta}, \hat{b}_{\xi,\eta})^T$ and $(\hat{r}_{\xi',\eta'}, \hat{b}_{\xi',\eta'})^T$ are substantially eliminated by the associated lowpass filters.

The product $\hat{I}_{2h}^{h}\hat{I}_h^{2h}$ can be expressed as

$$\hat{I}_{2h}^{h}\hat{I}_h^{2h} = \frac{1}{8} \begin{bmatrix} \hat{F}_{11} & \hat{F}_{12} \\ \hat{F}_{21} & \hat{F}_{22} \end{bmatrix}, \tag{5.5}$$

where

$$\hat{F}_{11} = \begin{bmatrix} (1+\beta)^2 & (1+\beta)(1+\tilde{\beta}) \\ (1+\beta)(1+\tilde{\beta}) & (1+\tilde{\beta})^2 \end{bmatrix}, \quad \hat{F}_{22} = \begin{bmatrix} 4\alpha^2 & 4\alpha\tilde{\alpha} \\ 4\alpha\tilde{\alpha} & 4\tilde{\alpha}^2 \end{bmatrix},$$

$$\hat{F}_{12} = \hat{F}_{21}^T = \begin{bmatrix} 2\alpha(1+\beta) & 2\tilde{\alpha}(1+\beta) \\ 2\alpha(1+\tilde{\beta}) & 2\tilde{\alpha}(1+\tilde{\beta}) \end{bmatrix}.$$

Therefore, from (5.2), (5.4), and (5.5), we obtain the frequency domain matrix for the coarse-grid correction operator

$$\hat{K}_h^{2h} = \begin{bmatrix} \hat{K}_{11} & \hat{K}_{12} \\ \hat{K}_{21} & \hat{K}_{22} \end{bmatrix}, \tag{5.6}$$

where

$$\hat{K}_{11} = I - \tfrac{1}{4\delta}[\hat{F}_{12}\hat{J} - \hat{F}_{11}], \quad \hat{K}_{22} = I - \tfrac{1}{4\delta}[\hat{F}_{21}\hat{J} - \hat{F}_{22}],$$
$$\hat{K}_{12} = -\tfrac{1}{4\delta}[\hat{F}_{11}\hat{J} - \hat{F}_{12}], \qquad \hat{K}_{21} = -\tfrac{1}{4\delta}[\hat{F}_{22}\hat{J} - \hat{F}_{21}].$$

The remaining task is to combine results of (5.3) and (5.6) so that the spectral radius $\rho(\hat{T}_h^{2h})$ can be determined.

5.3. Two-to-one wave reduction. The analytical determination of the eigenvalues of the two-grid iteration matrix $\hat{T}_h^{2h}$ is, in general, a difficult task since it is a 4×4 matrix. However, if the red/black Gauss-Seidel smoother is used, the whole process is greatly simplified. When the first partial step of the red/black Gauss-Seidel iteration, i.e., the Jacobi iteration at red points, is performed, the values of the red points are updated by the values of their neighboring black points and their original values are totally discarded. As a consequence, the computational process that follows is only determined by the initial values of the black points. This is clearly indicated by the first two zero columns in (5.3).

For the two-grid iteration process (5.2), let us temporarily consider the special case $(\nu_1, \nu_2) = (1, 0)$. For such a simple case, we find that

$$\hat{T}_h^{2h} = \begin{bmatrix} 0 & \hat{K}_{11}\hat{J} + \hat{K}_{12}\hat{J}^2 \\ 0 & \hat{K}_{21}\hat{J} + \hat{K}_{22}\hat{J}^2 \end{bmatrix}, \tag{5.7}$$

and the spectral radius of $\hat{T}_h^{2h}$ is

$$\rho(\hat{T}_h^{2h}) = \rho(\hat{K}_{21}\hat{J} + \hat{K}_{22}\hat{J}^2). \tag{5.8}$$

The two-to-one color reduction is mathematically clear from equations (5.7) and (5.8), namely, that (5.8) involves the evolution of black waves only. We can interpret its corresponding physical mechanism as follows. The two-grid iteration process $\hat{T}_h^{2h}$ consists of two processes

$$\hat{T}_{12} = \hat{K}_{11}\hat{J} + \hat{K}_{12}\hat{J}^2, \qquad \hat{T}_{22} = \hat{K}_{21}\hat{J} + \hat{K}_{22}\hat{J}^2,$$

which describe the evolution from $(b_{\xi,\eta}, -b_{\xi',\eta'})^T$ to $(r_{\xi,\eta}, -r_{\xi',\eta'})^T$ and $(b_{\xi,\eta}, -b_{\xi',\eta'})^T$, respectively. Since the m-fold repetition of $\hat{T}_h^{2h}$ gives

$$[(\hat{T}_h^{2h})^m]_{12} = \hat{T}_{12}\hat{T}_{22}^{m-1}, \qquad [(\hat{T}_h^{2h})^m]_{22} = \hat{T}_{22}^m,$$

the convergence of the two-grid method depends entirely on the process $\hat{T}_{22}$. Hence, the two-color two-grid iteration process (5.2) can be equivalently characterized by the black-wave two-grid iteration process.

For general (ν_1, ν_2), since

$$\rho[\hat{T}_h^{2h}(\nu_1, \nu_2)] = \rho(\hat{S}_{\text{RBGS}}^{\nu_2}\hat{K}_h^{2h}\hat{S}_{\text{RBGS}}^{\nu_1}) = \rho(\hat{K}_h^{2h}\hat{S}_{\text{RBGS}}^{\nu_1+\nu_2}),$$

where the last equality comes from the fact $\rho(AB) = \rho(BA)$, we can derive that

$$\rho[\hat{T}_h^{2h}(\nu_1, \nu_2)] = \rho(\hat{K}_{21}\hat{J}^{2\nu-1} + \hat{K}_{22}\hat{J}^{2\nu}),$$

where $\nu = \nu_1 + \nu_2$. Let us examine the matrix

$$\hat{T}_{\text{eq}} = \hat{K}_{21}\hat{J}^{2\nu-1} + \hat{K}_{22}\hat{J}^{2\nu},$$

which represents a one-color two-grid iteration process and, due to (5.6), can be expressed as

$$\hat{T}_{\text{eq}} = \hat{J}\hat{K}_{\text{eq}}\hat{J}^{2\nu-1},$$

where

$$\begin{aligned}\hat{K}_{\text{eq}} &= I - \frac{1}{4\delta}\hat{J}^{-1}\hat{F}_{21}(\hat{J}^2 - I)\\ &= \begin{bmatrix} 1 - \frac{(1+\beta)(\alpha^2-1)}{2\delta} & -\frac{(1+\tilde{\beta})(\tilde{\alpha}^2-1)}{2\delta} \\ -\frac{(1+\beta)(\alpha^2-1)}{2\delta} & 1 - \frac{(1+\tilde{\beta})(\tilde{\alpha}^2-1)}{2\delta} \end{bmatrix}\end{aligned}$$

is the equivalent one-color coarse-grid correction operator in the frequency domain. Since $\rho(\hat{J}\hat{K}_{\text{eq}}\hat{J}^{2\nu-1}) = \rho(\hat{K}_{\text{eq}}\hat{J}^{2\nu})$, we see that $\hat{J}^2$ can be viewed as the equivalent one-color smoother $\hat{S}_{\text{eq}}$, which corresponds to two Jacobi relaxation steps for the black component $b_{\xi,\eta}$.

5.4. The spectral radius result. The equivalent one-color two-grid iteration matrix can also be determined for the degenerate case $\xi = M/2$ or $\eta = M/2$ [21]. Then, the spectral radius of the two-grid iteration matrix can be found by solving

$$\rho(T_h^{2h}) = \max_{1 \le \xi, \eta \le \frac{M}{2}} \rho(\hat{T}_{\text{eq}}).$$

In [29], Stüben and Trottenberg reduced their analysis to the determination of the largest value among all the spectral radii of the frequency domain matrices $\hat{J}^{2\nu}\hat{K}_{\text{eq}}$. Since we have $\rho(\hat{J}\hat{K}_{\text{eq}}\hat{J}^{2\nu-1}) = \rho(\hat{J}^{2\nu}\hat{K}_{\text{eq}})$, these two different derivations lead to the same final result. A closed form of this quantity has been derived in [29, pp. 104-108], which is summarized as follows:

$$\rho[T_h^{2h}(\nu = \nu_1 + \nu_2)] = \begin{cases} \frac{1}{4}, & \nu = 1, \\ \frac{1}{2\nu}(\frac{\nu}{\nu+1})^{\nu+1}, & \nu \ge 2. \end{cases} \tag{5.9}$$

In the above expression, the maximum of $\rho(T_h^{2h})$ occurs at $(\xi\pi h, \eta\pi h) = (\pi/2, 0)$ or $(0, \pi/2)$ when $\nu = 1$ and at $(\cos^{-1}[(\nu/\nu+1)^{\frac{1}{2}}], \cos^{-1}[(\nu/\nu+1)^{\frac{1}{2}}])$ when $\nu \ge 2$.

By using the two-color Fourier analysis, we can clearly see why MG with the red/black Gauss-Seidel smoother has a good convergence behavior in spite of its poor smoothing property for the high frequency components. Through the red/black Gauss-Seidel iteration, the low and high frequency components are coupled and can be equivalently formulated as the coupling between red and black waves with the same low frequency component. It turns out that only the black wave plays a role and that the low frequency component of the black wave is solved by coarse-grid correction. Thus, we conclude that the very high frequency components, namely, those with (θ, ϕ) close to (π, π), are in fact corrected at the coarse grid rather than smoothed at the fine grid. Such an explanation is difficult to obtain using the analysis given by [29].

6. Convergence rate comparison for natural and red/black orderings.

6.1. SOR and SSOR methods. Fourier analysis has been used to analyze the naturally ordered SOR and SSOR iteration methods for the Poisson problem on a square with the *periodic* boundary conditions by Chan and Elman [13]. It is shown that the optimal relaxation parameters for both cases are the same,

$$\omega^*(\text{natural ordering; periodic b.c.}) = \frac{2}{1 + 2\sin(0.5\pi h)}, \tag{6.1}$$

and the corresponding spectral radii are

$$\rho_{\text{SOR}}(\text{natural ordering; periodic b.c.}) \approx 1 - 0.5\pi h, \tag{6.2}$$

$$\rho_{\text{SSOR}}(\text{natural ordering; periodic b.c.}) \approx 1 - \pi h. \tag{6.3}$$

For the model Dirichlet problem, Frankel derived a classical Fourier result for the SOR iteration with the natural ordering [18]. That is, the optimal relaxation parameter is

$$\omega^*(\text{natural ordering; Dirichlet b.c.}) = \frac{2}{1 + \sin \pi h}, \tag{6.4}$$

and the corresponding spectral radius is

$$\rho_{\text{SOR}}(\text{natural ordering; Dirichlet b.c.}) \approx 1 - 2\pi h. \tag{6.5}$$

This result was interpreted by LeVeque and Trefethen from a tilted-grid point of view [24]. Although there is no Fourier result of the naturally ordered SSOR iteration for the Dirichlet problem, it can be shown by matrix analysis that

$$\rho_{\text{SSOR}}(\text{natural ordering; Dirichlet b.c.}) \le 1 - \pi h, \tag{6.6}$$

and that the convergence rate is not sensitive to the choice of the relaxation parameter [19],[32]. Note that (6.1)-(6.3) agrees with (6.4)-(6.6) asymptotically except for the constant multiplying h in (6.2) and (6.5).

By comparing the above results with those in §3, we can clearly see that for the SOR iteration the red/black ordering does not effect the choice of the optimal relaxation parameters (cf. (3.5) and (6.4)) and the rate of convergence (cf. (3.6) and (6.5)). However, for the SSOR iteration, the situation changes drastically. If the red/black ordering is used, the acceleration due to the introduction of the relaxation parameter totally disappears (cf. (3.11), (6.3), and (6.6)).

6.2. Preconditioners. Chan and Elman also applied Fourier analysis to analyze the eigenstructures of the preconditioned system with the periodic boundary conditions and the natural ordering [13]. Their results are summarized as follows:

$$\kappa(Q_S^{-1}A)(\text{natural ordering; periodic b.c.}) = O(h^{-1}), \tag{6.7}$$

$$\kappa(Q_I^{-1}A)(\text{natural ordering; periodic b.c.}) = O(h^{-2}), \tag{6.8}$$

$$\kappa(Q_M^{-1}A)(\text{natural ordering; periodic b.c.}) = \begin{cases} O(h^{-2}) & c = 0 \\ O(h^{-1}) & c \neq 0 \end{cases} \tag{6.9}$$

where Q_S, Q_I, and Q_M denote the SSOR, ILU, and MILU preconditioning operators. Although no Fourier result for the naturally ordered Dirichlet problem is available,

TABLE 1
Comparison of convergence rates.

ordering	SOR	SSOR	PCG	MG
natural	$O(N^{1/2})$	$O(N^{1/2})$	$O(N^{1/4})$	$O(1)$
red/black	$O(N^{1/2})$	$O(N)$	$O(N^{1/2})$	$O(1)$

TABLE 2
Comparison of the spectral radii for the MG *method.*

ordering	$\nu = 1$	$\nu = 2$	$\nu = 3$
natural	1/2	1/4	1/8
red/black	1/4	2/27	27/512

these results agree with the known results for the Dirichlet case (see the references of [13]) and numerical experiments indicate that the eigenstructures for the periodic and Dirichlet cases behave in a very similar way [13].

By examining (4.7), (4.9), and (4.12), we see that the preconditioned system with the red/black ordering, in general, does not decrease the order of the condition number of the original Laplacian. In fact, the condition number is reduced approximately by a factor 4 for SSOR, ILU, and MILU preconditioners. In contrast, effective naturally ordered preconditioners such as SSOR and MILU can decrease the condition number of the Laplacian by an order of magnitude. Thus, as far as the convergence rate is concerned, a red/black preconditioned iterative method usually converges much slower than a naturally ordered preconditioned iterative method.

The condition number analysis of the red/black ordered preconditioners is consistent with the experimental results reported by Ashcraft and Grimes [5] and to the best of our knowledge, no such analysis has been reported before.

6.3. MG methods. So far, there is no exact Fourier result for the two-grid analysis of the model Dirichlet problem with natural ordering. However, a simplified local Fourier analysis that assumes ideal interpolation and restriction operators and ignores the boundary conditions has been performed by Brandt [10]. The smoothing rate μ of one lexicographical Gauss-Seidel relaxation is found to be 1/2 by such an analysis. When the total number ν of the smoothing iteration is small, we can roughly estimate the spectral radius of two-grid iteration matrix from the smoothing rate by

$$\rho_{\mathrm{MG}}(\text{natural ordering}) \approx \mu^\nu = (\frac{1}{2})^\nu. \tag{6.10}$$

Therefore, from (5.9) and (6.10), we see that the red/black Gauss-Seidel smoother has a better smoothing rate than that of the lexicographical Gauss-Seidel smoother.

6.4. Summary of comparison. We summarize the above comparison in Table 1, where each entry denotes the number of iterations required and N is the number of unknowns. The spectral radii of the MG method, which are calculated by (5.9) and (6.10), are also compared in Table 2.

7. Related work. Most research work on iterative algorithms with the multicolor ordering has been focused on the SOR method. To achieve the efficiency of the SOR iteration, the determination of the optimal relaxation parameter is crucial. However, except for a few simple cases such as the model Poisson problem, this is, in general, a difficult task. A local two-color Fourier analysis has been proposed by

Kuo, Levy, and Musicus [23] to design a local relaxation scheme that uses different relaxation parameters for different finite-difference equations associated with each grid point. The four-color SOR iteration applied to the 9-point Laplacian has been independently studied by Adams, LeVeque, and Young [3] and Kuo and Levy [22]. The technique used by Adams, LeVeque, and Young is to change the variable of iteration number to a new variable known as the "data flow time" defined by Adams and Jordan [2]. By using such a technique, the multicolor ordering scheme can be related to the natural ordering scheme and then analyzed by a modified Fourier analysis. In [22], Kuo and Levy used a four-color Fourier analysis to design a two-level SOR scheme that includes an outer block SOR iteration and an inner point SOR iteration. The four-color Fourier analysis is a natural generalization of the two-color Fourier analysis presented in this paper. In addition to the four-color ordering, O'Leary has considered several other ordering schemes for the 9-point Laplacian and has shown that the convergence rate of the SOR iteration with these orderings is no worse than that for the natural ordering [26].

8. Conclusions and extensions.

We conclude our study simply as follows. Although some algorithms such as the SOR and MG methods can be reordered to get more parallelism without sacrificing their convergence rates, some algorithms such as the SSOR and preconditioned iterative methods do have a trade-off in achieving more parallelism and faster convergence.

A natural question that arises from this research work is: what is the "intrinsic property" of these algorithms that makes them behave so differently with respect to the reordering? A better understanding of this fundamental issue should help us to know more about parallel computation and its limitation. The question of the poor performance of the red/black SSOR, ILU, and MILU preconditioners can be partly answered by the observation that at each iteration the red/black preconditioners use only local information, whereas the naturally ordered preconditioners do make use of some global information.

The preconditioned iterative methods such as the PCG method are among one of the most effective methods for solving elliptic PDEs in a sequential machine. However, since effective preconditioners such as the naturally ordered SSOR and MILU schemes cannot be easily parallelized, they are not as attractive for parallel computers. It is an interesting and important research topic to find preconditioners that are easily parallelizable and give satisfactory convergence rates.

Acknowledgment. The authors are grateful to Professor Stanley C. Eisenstat for his helpful comments regarding the presentation in §2.

REFERENCES

[1] L. M. ADAMS, *Iterative algorithms for large sparse linear systems on parallel computers*, Ph.D. thesis, Department of Applied Mathematics and Computer Science, University of Virginia, Charlottesville, VA, 1982; also published as NASA CR-166027, NASA Langley Research Center, Hampton, VA.

[2] L. M. ADAMS AND H. F. JORDAN, *Is* SOR *color-blind?* SIAM J. Sci. Statist. Comput., 7 (1986), pp. 490-506.

[3] L. ADAMS, R. J. LEVEQUE, AND D. M. YOUNG, *Analysis of the* SOR *iteration for the* 9*-point Laplacian*, SIAM J. Numer. Anal., 25 (1988), pp. 1156-1180.

[4] L. M. ADAMS AND J. M. ORTEGA, *A multi-color* SOR *method for parallel computation*, ICASE report 82-9, ICASE-NASA Langley Research Center, Hampton, VA, 1982.

[5] C. C. ASHCRAFT AND R. G. GRIMES, *On vectorizing incomplete factorization and* SSOR *preconditioners*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 122-151.

[6] O. AXELSSON, *A generalized* SSOR *method*, BIT, 13 (1972), pp. 443-467.

[7] ———, *Solution of linear systems of equations: Iterative methods*, in Sparse Matrix Techniques, V. A. Barker, ed., Springer-Verlag, Berlin, New York, 1977, pp. 1-51.

[8] ———, *A survey of vectorizable preconditioning methods for large scale finite element matrix problem*, Report CNA-190, Center for Numerical Analysis, University of Texas, Austin, TX, 1984.

[9] ———, *A survey of preconditioned iterative methods for linear systems of algebraic equations*, BIT, 25 (1985), pp. 166-187.

[10] A. BRANDT, *Multi-level adaptive solutions to boundary-value problems*, Math. Comp., 31 (1977), pp. 333-390.

[11] ———, *Multigrid solvers on parallel computers*, in Elliptic Problem Solvers, M. H. Schultz, ed., Academic Press, New York, 1981, pp. 39-83.

[12] T. F. CHAN, *Analysis of preconditioners for domain decomposition*, SIAM J. Numer. Anal., 24 (1987), pp. 382-390.

[13] T. F. CHAN AND H. C. ELMAN, *Fourier analysis of iterative methods for elliptic problems*, SIAM Rev., 31 (1989), pp. 20-49.

[14] T. F. CHAN AND D. C. RESASCO, *A framework for the analysis and construction of domain decomposition preconditioners*, in First International Symposium on Domain Decomposition Methods for Partial Differential Equations, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988, pp. 217-230.

[15] R. E. CROCHIERE AND L. R. RABINER, *Multirate Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[16] G. DAHLQUIST AND A. BJORCK, *Numerical Methods*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[17] T. DUPONT, R. P. KENDALL, AND H. H. RACHFORD JR., *An approximate factorization procedure for solving self-adjoint difference equations*, SIAM J. Numer. Anal., 5 (1968), pp. 559-573.

[18] S. P. FRANKEL, *Convergence rates of iterative treatments of partial differential equations*, Math. Tables Aids Comput., 4 (1950), pp. 65-75.

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

[20] C.-C. J. KUO, *Discretization and solution of elliptic* PDEs: *A transform domain approach*, Ph.D. thesis, Report LIDS-TH-1687, Laboratory for Information and Decision Systems, MIT, Cambridge, MA, August 1987.

[21] C.-C. J. KUO AND B. C. LEVY, *Two-color Fourier analysis of the multigrid method with red/black Gauss-Seidel smoothing*, Appl. Math. Comput., 29 (1989), pp. 69-87.

[22] ———, *A two-level four-color* SOR *method*, SIAM J. Numer. Anal., 26 (1989), pp. 129-151.

[23] C.-C. J. KUO, B. C. LEVY, AND B. R. MUSICUS, *A local relaxation method for solving elliptic* PDE*s on mesh-connected arrays*, SIAM J. Sci. Stat. Comput., 8 (1987), pp. 550-573.

[24] R. J. LEVEQUE AND L. N. TREFETHEN, *Fourier analysis of the* SOR *iteration*, IMA J. Numer. Anal., 8 (1988), pp. 273-279.

[25] 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.

[26] D. P. O'LEARY, *Ordering schemes for parallel processing of certain mesh problems*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 620-632.

[27] J. M. ORTEGA AND R. G. VOIGT, *Solution of partial differential equations on vector and parallel computers*, SIAM Rev., 27 (1985), pp. 149-240.

[28] Y. SAAD AND M. H. SCHULTZ, *Parallel implementations of preconditioned conjugate gradient methods*, Research Report YALEU/DCS/RR-425, Yale University, New Haven, CT, 1985.

[29] K. STÜBEN AND U. TROTTENBERG, *Multigrid methods: Fundamental algorithms, model problem analysis, and applications*, in Multigrid Methods, W. Hackbusch and U. Trottenberg, eds., Springer-Verlag, New York, 1982, pp. 1-176.

[30] D. M. YOUNG, *Iterative methods for solving partial differential equations of elliptic type*, Ph.D. thesis, Harvard University, Cambridge, MA, 1950.

[31] ———, *Iterative Solution of Large Linear Systems*, Academic Press, New York, 1971.

[32] ———, *On the accelerated* SSOR *method for solving large linear systems*, Adv. in Math., 23 (1977), pp. 215-271.

[33] D. YOUNG, T. OPPE, D. KINCAID, AND L. HAYES, *On the use of vector computers for solving large sparse linear systems*, Report CNA-199, Center for Numerical Analysis, University of Texas, Austin, TX, 1984.

SIAM J. SCI. STAT. COMPUT. Vol. 11, No. 4, pp. 794–803, July 1990

TIMELY COMMUNICATIONS

Under the "timely communcations" policy for the SIAM Journal on Scientific and Statistical Computing, papers that have significant timely content and do not exceed five pages automatically will be considered for a separate section of the journal with an accelerated reviewing process. It will be possible for the note to appear approximately six months after the date of acceptance.

A NOTE ON THE EFFICIENCY OF DOMAIN DECOMPOSED INCOMPLETE FACTORIZATIONS [*]

TONY F. CHAN[†] AND DANNY GOOVAERTS[‡]

Abstract. In the past several years, domain decomposition has been a very popular topic, motivated by the ease of parallelization. However, the question of whether it is better than parallelizing some standard sequential methods has seldom been directly addressed.

In this paper it is shown, with some numerical examples, that the answer is affirmative in the case of iterative solutions of elliptic problems by preconditioned conjugate gradient iteration. Specifically shown is how to construct effective incomplete factorization preconditioners based on the domain decomposition principle. In addition to having all the advantages of domain decomposition, this also results in better convergence rates than the analogous preconditioners on the whole domain.

Key words. parallel algorithms, domain decomposition, partial differential equations, preconditioned conjugate gradient

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

1. Introduction.

In the past several years, domain decomposition methods for solving elliptic partial differential equations have attracted much attention. The main impulse for the enormous interest in these methods has come from the arrival of parallel computers. Besides the ease of parallelization, domain decomposition allows us to treat complex geometries or to isolate singular parts of the domain.

In the majority of domain decomposition methods, the matching of the solution on the subdomains to an overall solution is done by an iterative process. A large class is based on the preconditioned conjugate gradient method (PCG) for solving the reduced equations on the interfaces between the subdomains. The efficiency of these methods is determined by the preconditioner used. This approach involves a solve on each subdomain in each iteration step, and the cost could be expensive if the number of iterations is not kept at a minimum. Therefore, doubts have been raised on the efficiency of these methods as compared to a parallelization of traditional preconditioned conjugate gradient iterations on the whole domain.

[*] Received by the editors June 24, 1988; accepted for publication (in revised form) January 11, 1989. This research was supported in part by the Department of Energy under contract DE-FG03-87ER25037, by the National Science Foundation under contract NSF-DMS87-14612 and by the Army Research Office under contract DAAL03-88-K-0085.

[†] Department of Mathematics, University of California, Los Angeles, California 90024 (chan@math.ucla.edu or na.tchan@na-net.stanford.edu).

[‡] Department of Computer Science, Katholieke Universiteit, Leuven, Celestijnenlaan 200 A, 3030 Leuven, Belgium (danny@cs.kuleuven.ac.be or na.goovaerts@na-net.stanford.edu). This research was done during a visit to the Department of Mathematics, University of California, Los Angeles, California, January – June 1988. This author is holder of a Senior Research Assistantship of the National Fund for Scientific Research Belgium (NFWO/FNRS). In addition, his visit to the University of California at Los Angeles was sponsored by a travel grant of the National Fund for Scientific Research Belgium (NFWO/FNRS).

One aspect that has generally been ignored is the gain in sequential computational complexity that domain decomposition can yield as a divide-and-conquer technique. When the work for solving a problem grows more than linearly with its size, splitting it up in two subproblems of half the size will yield a faster method *provided* that the subsolutions can be efficiently combined to obtain the solution of the original problem. In this paper, we show that this can be exploited for incomplete factorization preconditioners.

Specifically, in §2, given a method for constructing a preconditioner $\widehat{M}$ on the whole domain (such as the ILU- and MILU-type methods, which give a condition number for the preconditioned system of $O(h^{-2})$, $O(h^{-1})$, respectively), we show how to construct a domain decomposed preconditioner based on applying the same method in the subdomains and using an appropriately chosen preconditioner for the interface. We hereby stress the importance of the preconditioner on the interface, for a badly chosen one can adversely affect the overall convergence rate. One that we have found to be very successful is the boundary probe preconditioner [7]. This domain decomposed preconditioner for the original domain can yield a faster convergence rate with roughly the same operation count per iteration step, as compared to traditional preconditioners.

In §3 we stress the importance of an appropriate factorization of the subdomain preconditioners in order to minimize the number of arithmetic operations per iteration step. In §4 we show some numerical experiments to illustrate our main points.

Another instance of where a parallel algorithm, when executed sequentially, turns out to be better than the traditional sequential algorithms is the parallel method for solving the symmetric eigenvalue problem as proposed in [12].

2. Domain decomposed preconditioners.

We formulate this approach for the simplest case of a domain Ω split into two subdomains Ω_1 and Ω_2 sharing the interface Γ.

Consider the problem:

$$\begin{aligned} Lu &= f \quad \text{on } \Omega, \\ u &= u_b \quad \text{on } \partial\Omega, \end{aligned}$$

where L is a linear second-order elliptic operator.

Given a preconditioning method on Ω, we would like to construct a preconditioner that can be inverted in a domain decomposed way. To do this, we order the unknowns for the internal points of the subdomains first and those on the interface Γ last. Then the discrete solution vector $u = (u_1, u_2, u_3)^T$ satisfies the linear system:

$$Au = \begin{pmatrix} A_{11} & 0 & A_{13} \\ 0 & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix}. \tag{1}$$

The matrix A can be factored in block form:

$$A = \begin{pmatrix} A_{11} & & \\ & A_{22} & \\ A_{31} & A_{32} & S \end{pmatrix} \begin{pmatrix} I & & A_{11}^{-1}A_{13} \\ & I & A_{22}^{-1}A_{23} \\ & & I \end{pmatrix}, \tag{2}$$

where

$$S = A_{33} - A_{31}A_{11}^{-1}A_{13} - A_{32}A_{22}^{-1}A_{23}.$$

The matrix S is the Schur complement of A_{33} in the matrix A. It corresponds to the reduction of the operator L on Ω to an operator on the internal boundary Γ. Suppose we apply the preconditioning method to each subdomain Ω_i and obtain B_{ii}. One way to derive a preconditioner for A is by replacing A_{ii} in (2) by the approximations B_{ii} and replacing the Schur complement by a preconditioner M.

We therefore arrive at the following preconditioner:

$$\widetilde{M} = \begin{pmatrix} B_{11} & & \\ & B_{22} & \\ A_{31} & A_{32} & M \end{pmatrix} \begin{pmatrix} I & & B_{11}^{-1}A_{13} \\ & I & B_{22}^{-1}A_{23} \\ & & I \end{pmatrix}. \tag{3}$$

An advantage of the algebraical approach as expressed by (3) is that it allows an easy combination of preconditioners of any kind for the elliptic problems on the subdomains, as, e.g., incomplete factorizations or a number of multigrid cycles. However, care must be taken in the scaling of the preconditioners on the subdomains and on the interface to the unchanged off-diagonal blocks. The rate of convergence of the preconditioned conjugate gradient method is determined by the eigenvalue spectrum of $\widetilde{M}^{-1}A$ [10], which is given by:

$$\begin{pmatrix} B_{11}^{-1}A_{11} & 0 & 0 \\ 0 & B_{22}^{-1}A_{22} & 0 \\ 0 & 0 & M^{-1}\tilde{S} \end{pmatrix} - \begin{pmatrix} B_{11}^{-1}A_{13}M^{-1}A_{31}P_1 & B_{11}^{-1}A_{13}M^{-1}A_{32}P_2 & -B_{11}^{-1}A_{13}\left(I - M^{-1}\tilde{S}\right) \\ B_{22}^{-1}A_{23}M^{-1}A_{31}P_1 & B_{22}^{-1}A_{23}M^{-1}A_{32}P_2 & -B_{22}^{-1}A_{23}\left(I - M^{-1}\tilde{S}\right) \\ -M^{-1}A_{31}B_{11}^{-1}A_{13} & -M^{-1}A_{32}B_{22}^{-1}A_{23} & 0 \end{pmatrix}, \tag{4}$$

with

$$P_i = \left(I - B_{ii}^{-1}A_{ii}\right) \tag{5}$$

and

$$\tilde{S} = A_{33} - A_{31}B_{11}^{-1}A_{13} - A_{32}B_{22}^{-1}A_{23}. \tag{6}$$

From (4) and (5) we see that if the $B_{ii}^{-1}A_{ii}$, $i = 1,2$, and $M^{-1}\tilde{S}$, with $\tilde{S}$ given by (6), are close to the unity matrix then the eigenvalue spectrum of $\widetilde{M}^{-1}A$ is a perturbation of the spectra of $B_{ii}^{-1}A_{ii}$, $i = 1,2$ and of $M^{-1}\tilde{S}$. Therefore it is not enough that the condition number of $B_{ii}^{-1}A_{ii}$ is close to one, but the eigenvalues of $B_{ii}^{-1}A_{ii}$ must be clustered around one. For many commonly used preconditioners (e.g., ILU or multigrid preconditioners) this is indeed the case. For an extensive theoretical study of the eigenvalue spectrum, we refer to [16] and [17].

For M we can take any of the preconditioners for the Schur complement that were proposed in the literature [13], [15], [2], [6], [7], [1], [18]. We have found that a good overall preconditioner for S for general second-order partial differential equations is the boundary probe preconditioner [7]. Several experiments with this preconditioner have been performed [18]. The main motivation for this approach is the observation that, in the case of the Laplace operator, the elements of the matrix S decay rapidly away from the main diagonal [15]. It is therefore reasonable to consider a k-diagonal approximation to S. However, it would not be efficient to calculate the elements of S in order to do this.

Instead, as proposed in [7], a $(2k+1)$-diagonal approximation to S can be constructed by multiplying S by $2k+1$ "probing" vectors $v_j, j = 1, \cdots, 2k+1$. The idea is motivated by sparse Jacobian evaluation techniques [11]. For the case $k = 0$ and $k = 1$ the probing vectors are the following:

$$\begin{aligned} k=0: v_1 &= (1,1,1,1,1,1,1,\cdots)^T, \\ k=1: v_1 &= (1,0,0,1,0,0,1,\cdots)^T, \\ v_2 &= (0,1,0,0,1,0,0,\cdots)^T, \\ v_3 &= (0,0,1,0,0,1,0,\cdots)^T. \end{aligned}$$

The case $k = 0$ corresponds to a scaling of each row of the matrix S by the sum of the elements of the row. For $k = 1$, if S were indeed tridiagonal, all of its elements would be found in the vectors Sv_j, $j = 1, 2, 3$.

This approach for probing a matrix is valid for any matrix S and will yield a good banded approximation, provided that the elements of S decay away from the main diagonal. In the case of the Schur complement this is inherently related to the fact that the operator S is predominantly local. If the discretization stencil used extends over one gridline in each direction, the most important coupling of a gridpoint on the interface will be with its immediate neighbours, and a tridiagonal approximation for S will suffice.

The probing technique asks for $2k+1$ products Sv_j. This implies $(2k+1)$ solves on each subdomain. However, as indicated by (4), it suffices to have a preconditioner for $\tilde{S}$ given by (6). Since $\tilde{S}$ approximates S, it will also have the property that it is dominantly local. When the matrices A_{ii} are replaced by approximate factorizations, which correspond to more local operators, the diagonals of $\tilde{S}$ will decay even more. So the probing technique can be applied here also. However, the product $\tilde{S}v_j$ now only involves approximate solves and thus can be performed cheaply. Another probing technique used to construct a sparse approximation to the Schur complement is proposed in [1].

Preconditioners of this kind were first proposed by Bramble, Pasciak, and Schatz [5], [4]. They are constructed by replacing the bilinear form of the weak formulation of the differential equation by a spectrally equivalent form that can be inverted in a domain decomposed way. Algebraically it results in a preconditioner of the form (3) where the off-diagonal blocks are also replaced. As subdomain preconditioners, constant coefficient or separable approximations are used. Numerical experiments with these preconditioners are reported in [4] and in the survey paper by Keyes and Gropp [18]. This method is also studied in [24]. Preconditioners of the form (3), with the Neumann-Dirichlet preconditioner [2] and with the trace preconditioner [13] are studied in [3]. In [22] and [21] Meurant proposes preconditioners of the form (3) derived from block preconditioners. The interface preconditioner here follows naturally from the recursive construction of sparse approximations to the Schur complements as they arrive in the block Cholesky decomposition of the discretization matrices.

3. Minimizing the operation count per iteration step. In each step of a preconditioned conjugate gradient iteration, the following system must be solved:

$$\widetilde{M} z^k = r^k. \tag{7}$$

Straightforward implementation of this step using the block LU factorization (3) involves two solves on each subdomain with the matrix B_{ii} in each iteration step, one in the forward elimination step and one in the backsubstitution. This implies that the work per iteration step roughly doubles as compared to a PCG iteration with the same preconditioner on the whole domain. This is a relatively high price for a domain decomposed method to pay in the sequential case. This increase in work can only be overcome by a reduction of the number of iteration steps by a factor of two, which is not achievable for many problems. In the parallel case, to be competitive, the gain from ease of parallelization must compensate for this factor when compared with a straigthforward parallelization of a traditional PCG method for the whole domain [19].

Here we describe a general technique to save this factor of two by an appropriate factorization and an appropriate ordering of the unknowns when the matrices B_{ii} can be expressed in an LU factorization:

$$B_{ii} = L_i U_i.$$

This then gives the following factored form for the preconditioner $\widetilde{M}$:

$$\widetilde{M} = \begin{pmatrix} L_1 & & \\ & L_2 & \\ A_{31}U_1^{-1} & A_{32}U_2^{-1} & M \end{pmatrix} \begin{pmatrix} U_1 & & L_1^{-1}A_{13} \\ & U_2 & L_2^{-1}A_{23} \\ & & I \end{pmatrix}. \tag{8}$$

Written this way, the solution of (7) still takes four solves with the matrices L_i and four with U_i, or two solves on each subdomain. However, the product $A_{3i}U_i^{-1}y_i$ involves only a few components of the vectors $U_i^{-1}y_i$. More specifically, for a five-point stencil, we only need the components of $U_i^{-1}y_i$ adjacent to the gridpoints on the internal interface.

Ordering the unknowns linewise in the direction of the internal edge, with the lines ordered going towards the interface, we obtain that the product $A_{3i}U_i^{-1}y_i$ requires only the bottom right-hand corner of U_i^{-1}. The rest of the solution in the interior is not needed. An analogous assertion holds for the product $L_i^{-1}A_{i3}z_3^k$. When the domain is split into more than two subdomains, it is not clear how this can be done.

A similar situation also occurs for the domain decomposed fast Poisson solver on a rectangle [8], [9]. Similar techniques to save the factor of two were used by Meurant in the construction of domain decomposed preconditioners based on block preconditioners [22], [21], [23] using alternating LU and UL factored preconditioners on the subdomains. In the two subdomain case, this corresponds to the technique we propose.

4. Numerical experiments.

In this section we present some numerical experiments that will illustrate the point of view we made earlier. We consider second-order self-adjoint differential operators:

$$Lu = \frac{\partial}{\partial x}\left(a(x,y)\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(b(x,y)\frac{\partial u}{\partial y}\right) + c(x,y)u.$$

The equation is discretized using a standard five-point second-order discretization stencil [14] on a equidistant grid. In our examples, the domain Ω is the unit square and is divided in two rectangular strips:

$$\Omega_1 = [0,1] \times [0,0.5], \quad \Omega_2 = [0,1] \times [0.5,1].$$

Taking n internal gridlines in each direction with gridsize $h = 1/(n+1)$, the system for the whole domain is of the order $n \times n$ and for the subdomains of the order $n \times n/2$. The interface has n internal points.

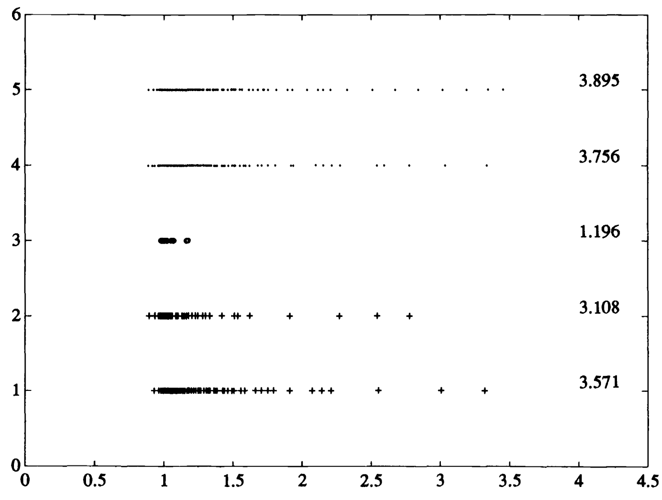


FIG. 1

For the preconditioner on the interface, we take $M = M_P(\tilde{S}, 1)$, the boundary probe preconditioner on $\tilde{S}$ with $k = 1$ (tridiagonal approximation). As approximate factorizations on the subdomains we use the ILU preconditioner [20], denoted by $B_{ii} \leftarrow$ ILU, or the MILU preconditioner with $\alpha = h^2$ [14], denoted by $B_{ii} \leftarrow$ MILU. PCG iteration with the ILU preconditioner has a computational complexity of $O(h^{-3})$ and with the MILU preconditioner of $O(h^{-2.5})$, so we can expect that the divide-and-conquer effect in domain decomposition will yield a faster method.

Example 1. In this example, we illustrate the relation between the eigenvalue spectrum of $\widetilde{M}^{-1}A$ and the eigenvalue spectra of $B_{11}^{-1}A_{11}$, $B_{22}^{-1}A_{22}$, and $M^{-1}\tilde{S}$. The operator, the gridsize, and the preconditioners for this example are summarized in the following table.

$$\begin{array}{llll} a = e^{5xy}, & h = 1/16, & M = M_P, & B_{ii} \leftarrow \text{MILU} \\ b = e^{-5xy}, & n = 15, & & \\ c = \frac{1}{1+x+y} & & & \end{array}$$

The eigenspectra are plotted in Fig. 1.

(1) $B_{11}^{-1}A_{11}$,
(2) $B_{22}^{-1}A_{22}$,
(3) $M_P^{-1}\tilde{S}$,
(4) $\widetilde{M}^{-1}A$,
(5) $\widehat{M}^{-1}A$.

$\widehat{M}$ is the MILU preconditioner on the whole domain Ω, with the natural ordering of the unknowns. The numbers to the right of the eigenspectra are the condition numbers, $\kappa = \lambda_{\max}/\lambda_{\min}$, of these matrices. From line 3, we immediately see that the probing preconditioner yields a very good approximation to $\tilde{S}$. As indicated by (3) we see that the eigenspectrum of $\widetilde{M}^{-1}A$ is a small perturbation of the union of the eigenspectra of $B_{11}^{-1}A_{11}$, $B_{22}^{-1}A_{22}$, and $\widetilde{M}^{-1}\tilde{S}$. The spectrum of $M_P^{-1}\tilde{S}$ is tightly clustered around 1 and lies fully within the spectra of $B_{11}^{-1}A_{11}$ and $B_{22}^{-1}A_{22}$. This leads to a good spectrum for $\widetilde{M}^{-1}A$ with a condition number that is smaller than for the sequential preconditioner.

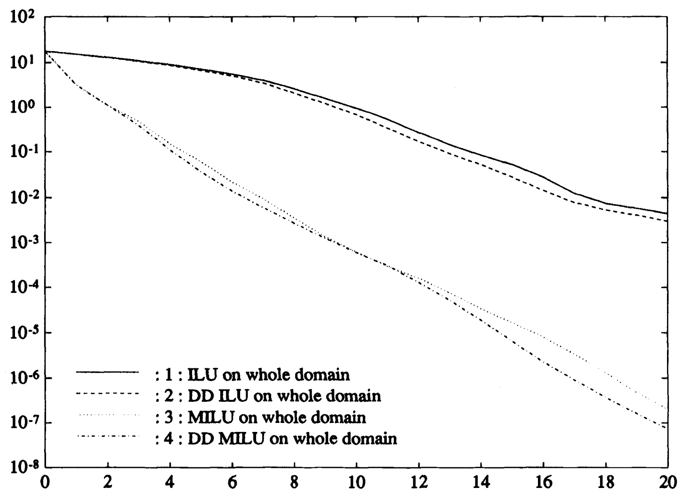


FIG. 2

Example 2. In this example, we show that our domain decomposed preconditioner yields a faster convergence than an analogous preconditioner applied to the whole domain for smooth problems.

The problem is the following:

$$\begin{array}{llll} a = -e^{xy}, & h = 1/32, & M = M_P, & B_{ii} \leftarrow \text{ILU} \\ b = -e^{-xy}, & n = 31, & & B_{ii} \leftarrow \text{MILU} \\ c = \frac{1}{1+x+y} & & & \end{array}$$

The right-hand side of the discrete equations is taken such that the exact discrete solution u_e satisfies

$$(u_e)_{ij} = x_i.$$

As a starting guess we used $u^0_{ij} = 1$. In Fig. 2 we plot $e^k = \left\|u^k - u_e\right\|_2$ versus the iteration count k. We compare the following cases:

(1) $\widehat{M} \leftarrow \text{ILU}$,

(2) $\widetilde{M}$, with $B_{ii} \leftarrow \text{ILU}$,

(3) $\widehat{M} \leftarrow \text{MILU}$,

(4) $\widetilde{M}$, with $B_{ii} \leftarrow \text{MILU}$,

where $\widehat{M}$ denotes the sequential preconditioner on the whole domain.

For both choices for the preconditioner on the subdomains, the domain decomposed preconditioner has a slightly faster rate of convergence. Several other experiments that we have done with other operators have all confirmed this point.

Example 3. In this example, we show a more dramatic gain, achieved for a problem with discontinuous coefficients.

The coefficients of the equation and the preconditioners are

$$\begin{array}{llll} a = -d(x,y), & h = 1/16, & M = M_P, & B_{ii} \leftarrow \text{ILU} \\ b = -d(x,y), & n = 15, & & \\ c = 0, & & & \end{array}$$

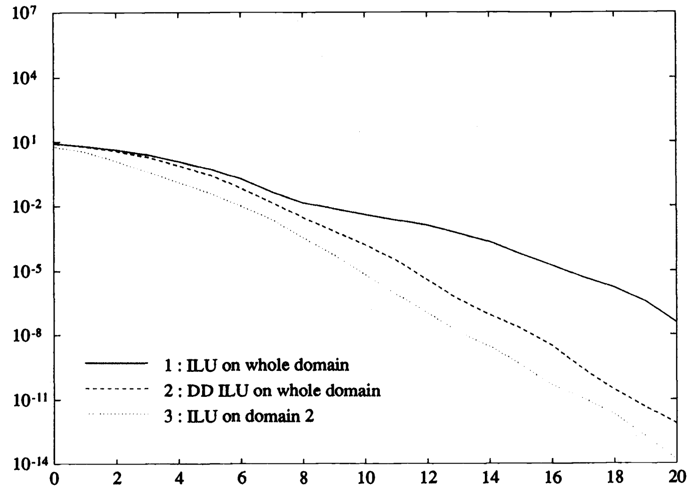


FIG. 3

with

$$\begin{aligned} d(x,y) &= 1000, & y &> 0.5, \\ d(x,y) &= 500.5, & y &= 0.5, \\ d(x,y) &= 1, & y &< 0.5. \end{aligned}$$

The exact discrete solution and the starting guess are as in Example 2. Figure 3 gives $e^k = \left\|u^k - u_e\right\|_2$ versus the iteration count k. We compare the following preconditioners:

(1) $\widehat{M} \leftarrow$ ILU,
(2) $\widetilde{M}$, with $B_{ii} \leftarrow$ ILU,
(3) ILU PCG on just one subdomain with the exact value of the solution on the interface.

For this problem, the domain decomposed preconditioner yields a much better convergence than the analogous preconditioner on the whole domain. Comparing it with the PCG iteration on just one subdomain, we see in the beginning of the iteration that it is somewhat slower than PCG on the subdomain. After some iterations, the rate of convergence is about the same. This illustrates clearly how domain decomposition allows us to exploit the smoother coefficients within each subdomain.

In [18] Keyes and Gropp also find that the Bramble-Pasciak-Schatz preconditioner with constant coefficient or separable preconditioners on the subdomains outperforms the corresponding preconditioner on the whole domain in number of iteration steps. The same observation holds for the domain decomposed block preconditioner INVkP as compared to the one domain version INV [22].

Acknowledgments. The second author expresses his profound appreciation for the hospitality and the inspiring environment provided by the Applied Mathematics Group and especially by Prof. T. Chan during the author's stay at the University of California at Los Angeles.

REFERENCES

[1] O. Axelsson and B. Polman, *Block preconditioning and domain decomposition methods, I*, Department of Mathematics Report 8735, Katholieke Universiteit Nijmegen, December 1987.

[2] P. E. Bjørstad and O. B. Widlund, *Iterative methods for the solution of elliptic problems on regions partitioned into substructures*, SIAM J. Numer. Anal., 23 (1986), pp. 1097–1120.

[3] C. Börgers, *The Neumann-Dirichlet domain decomposition method with inexact solvers on the subdomains*, Numer. Math., 55 (1989), pp. 123–136.

[4] J. H. Bramble, J. E. Pasciak, and A. H. Schatz, *The construction of preconditioners for elliptic problems by substructures*, Math. Comp., 47 (1986), pp. 103–134.

[5] ———, *An iterative method for elliptic problems on regions partitioned into substructures*, Math. Comp., 46 (1986), pp. 361–369.

[6] T. F. Chan, *Analysis of preconditioners for domain decomposition*, SIAM J. Numer. Anal., 24 (1987), pp. 382–390.

[7] T. F. Chan and D. Resasco, *A survey of preconditioners for domain decomposition*, in Proc. IV Coloquio de Matemáticas, Taller de Análisis Numérico y sus Aplicaciones, Taxco, Guerrero, Mexico, August 1985; Springer-Verlag, Berlin, New York, 1985.

[8] ———, *A domain decomposed fast Poisson solver on a rectangle*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s14–s26.

[9] ———, *Hypercube implementation of domain decomposed fast Poisson solvers*, in Proc. 2nd Conference on Hypercube Processors, M. Heath, ed., Knoxville, TN, August 1986; Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987, pp. 738–746.

[10] P. Concus, G. H. Golub, and D. O'Leary, *A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations*, in Sparse Matrix Computation, J. Bunch and D. Rose, eds., Academic Press, New York, 1976, pp. 309–322.

[11] A. R. Curtis, M. J. D. Powell, and J. K. Reid, *On the estimation of sparse Jacobian matrices*, J. Inst. Math. Appl., 13 (1974), pp. 117–119.

[12] J. Dongarra and D. Sorensen, *A fully parallel algorithm for the symmetric eigenvalue problem*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s139–s159.

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

[14] T. Dupont, R. Kendall, and H. H. Rachford, *An approximate factorization procedure for solving self-adjoint elliptic difference equations*, SIAM J. Numer. Anal., 5 (1968), pp. 559–573.

[15] G. H. Golub and D. Mayers, *The use of pre-conditioning over irregular regions*, in Proc. 6th Internat. Conference on Computational Methods in Science and Engineering, Versailles, France, December 12-16, 1983.

[16] D. Goovaerts, *Domain decomposition methods for elliptic partial differential equations*, Ph.D. thesis, Department of Computer Science, Katholieke Universiteit Leuven, Louvain, Belgium, 1990.

[17] D. Goovaerts, T. F. Chan, and R. Piessens, *The eigenvalue spectrum of domain decomposed preconditioners*, Tech. Report 126, Department of Computer Science, Katholieke Universiteit Leuven, Louvain, Belgium, December 1989; Appl. Numer. Math., Special Issue on Domain Decomposition Methods, to appear.

[18] D. E. Keyes and W. Gropp, *A comparison of domain decomposition techniques for elliptic partial differential equations*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s166–s202.

[19] ———, *Domain decomposition techniques for non symmetric systems of elliptic boundary value problems: Examples from* CFD, in Proc. 2nd International Symposium on Domain Decomposition Methods, T. F. Chan, R. Glowinsky, J. Périaux, and O. Widlund, eds., University of California, Los Angeles, CA, January 14-16; Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989, pp. 321–339.

[20] 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.

[21] G. Meurant, *Domain decomposition preconditioners for the conjugate gradient method*, Calcolo, 25 (1988), pp. 103–119.

[22] ———, *Domain decomposition versus block preconditioning*, in Proc. 1st International Symposium on Domain Decomposition Methods, R. Glowinsky, G. H. Golub, G. A. Meurant, and J. Périaux, eds., Paris, France, 1987; Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988, pp. 231–249.

[23] ———, *Domain decomposition methods for partial differential equations on parallel computers*,

in Vector and Parallel Computing, Issues in Applied Research and Development, J. Dongarra, I. Duff, P. Gaffney, and S. McKee, eds., Ellis Horwood Limited, Chichester, 1989, pp. 241–253.

[24] O. B. WIDLUND, *Iterative substructuring methods: Algorithms and theory for elliptic problems in the plane*, in Proc. 1st International Symposium on Domain Decomposition Methods, R. Glowinsky, G. H. Golub, G. A. Meurant, and J. Périaux, eds., Paris, France, 1987; Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988, pp. 113–128.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 4, pp. 804-809, July 1990

EXPERIENCE WITH A MATRIX NORM ESTIMATOR*

NICHOLAS J. HIGHAM†

Abstract. Fortran 77 codes for estimating the 1-norm of a real or complex matrix were presented by Higham in [*ACM Trans. Math. Software*, 14 (1988), pp. 381–396]. The codes have found use in various applications and have been adopted by two program libraries. Further observations about the norm estimation algorithm and experience in using it are reported here. In particular, an example is given where the algorithm requires nearly the maximum possible number of iterations.

Key words. matrix 1-norm, condition number estimation

AMS(MOS) subject classifications. primary 65F05, 65F35

1. Introduction. In [9] Higham presented two Fortran 77 codes for estimating the 1-norm of a real matrix or a complex matrix. The codes are in the NAG library at Mark 13 [13, Chap. F04], are being used for all the condition number estimation in LAPACK [3], and have been used with research or production codes by several workers [1], [2], [5], [6], [12]. Because of this wide use of the routines we feel it is worth reporting experience and insights accrued since the work in [9] was done. We describe here a new example of slow convergence of the norm estimation algorithm, point out that it is a special case of a more general iteration for estimating matrix norms, summarize the practical performance of the estimator, and describe several interesting applications.

First, we recall the original algorithm of Hager [7], on which the estimators in [9] are based. Our notation is as follows: e is the vector of all ones, e_j is the jth column of the identity matrix, and $\xi = \text{sign}(y)$ means $\xi_i = 1$ or -1 according as $y_i \geq 0$ or $y_i < 0$.

ALGORITHM 1. Given $B \in \mathbb{R}^{n\times n}$ this algorithm computes γ and $y = Bx$ such that $\gamma \leq \|B\|_1$ with $\|y\|_1/\|x\|_1 = \gamma$.

```
$x = e/n$
repeat
    $y = Bx$
    $\xi = \text{sign}(y)$
    $z = B^T\xi$
    if $\|z\|_\infty \leq z^Tx$
        $\gamma = \|y\|_1$
        quit
    end
    $x = e_j$, where $|z_j| = \|z\|_\infty$ (smallest such $j$)
end
```

An important feature of the algorithm is that it requires only a means for evaluating matrix-vector products Bx and $B^T\xi$—it does not require explicit access to the matrix B. Of course, if B were available explicitly we could simply compute $\|B\|_1 = \max_j \sum_i |b_{ij}|$.

* Received by the editors October 19, 1989; accepted for publication (in revised form) December 5, 1989.

† Department of Mathematics, University of Manchester, Manchester, M13 9PL, United Kingdom (na.nhigham@na-net.stanford.edu).

To derive Algorithm 1, Hager regards $\|B\|_1$ as the maximum of $F(x) = \|Bx\|_1$ over the unit ball S in the 1-norm and, since F is nondifferentiable at points x where Bx has a zero component, he applies an optimization technique based on subgradients. A subgradient of F at x is any vector g such that

$$F(y) \geq F(x) + g^T(y - x) \qquad \text{for all } y \in S.$$

See [7]–[9] for further details of the subgradient-based derivation. An alternative derivation relying on a simple heuristic argument is given in [9].

Hager notes that Algorithm 1 requires at most $n+1$ iterations, since $F(x) = \|y\|_1 = z^T x$ increases strictly on each stage and so each of the vertices e_j is visited at most once. He proves that the final vector x is a local maximizer for F as long as $y = Bx$ has no zero components.

We noticed recently that Algorithm 1 is a special case of an iteration investigated in [14] for estimating the mixed subordinate norm

$$(1) \qquad \|B\|_{\alpha,\beta} = \max_{x \neq 0} \frac{\|Bx\|_\beta}{\|x\|_\alpha}, \qquad B \in \mathbb{R}^{m \times n},$$

where $\|\cdot\|_\alpha$ and $\|\cdot\|_\beta$ are arbitrary vector norms. The iteration is

$$(2) \qquad x^{k+1} = h(z^k) = h\big(B^T g(Bx^k)\big),$$

where g and h are subgradients of the β-norm and the norm dual to the α-norm, respectively. With $\alpha = \beta = 1$ this iteration reduces to the one in Algorithm 1. In [14] it is proved that iteration (2) converges in a finite number of steps if one of the α- and β-norms is polyhedral (this class includes the 1- and ∞-norms, but not the 2-norm), but no properties of the limit point are determined. In [15] the special case where $\alpha = \infty$, $\beta = 1$, and B is symmetric positive definite is examined in detail. Unfortunately, the results in [14] and [15] do not seem to provide any new insight into Algorithm 1.

Two key aspects of the behaviour of Algorithm 1 are as follows.

(1) The estimates produced by the algorithm are frequently exact ($\gamma = \|B\|_1$), usually "acceptable" ($\gamma \geq \|B\|_1/10$), and sometimes poor ($\gamma < \|B\|_1/10$). Several families of matrices are known for which $\gamma/\|B\|_1$ can be arbitrarily small [8], [9].

(2) The algorithm almost always converges after at most four iterations [7]–[9].

Based on these, and other observations, the following changes to Algorithm 1 were made in [9]. We will refer to the modified algorithm as Algorithm EST.

Definition of estimate. To overcome most of the poor estimates, γ is redefined as

$$(3) \quad \gamma = \max\left\{\|y\|_1, \frac{\|w\|_1}{\|v\|_1}\right\}, \quad \text{where} \quad w = Bv, \quad v_i = (-1)^{i+1}\left(1 + \frac{i-1}{n-1}\right).$$

The vector v is considered likely to "pick out" any large elements of B in those cases where such elements fail to propagate through to y.

Convergence test. The algorithm is limited to a minimum of two and a maximum of five iterations. Also, convergence is declared after computing ξ if the new ξ is the same as the previous one (which signals that convergence will be obtained on the current iteration) *or* if the new $\|y\|_1$ is no larger than the previous one. The latter event can happen only in finite precision arithmetic and indicates that a vertex e_j is being revisited—the onset of "cycling."

In the following, all comments about Algorithm EST apply also to a version applicable to complex matrices [9], whose main difference from Algorithm EST is that it does not have the test for repeated ξ vectors.

Experience subsequent to the development of Algorithm EST has confirmed that the above modifications work well, and we are not aware of any way to improve the algorithm's performance. We have applied Algorithm EST to the test matrices in the collection [10], for values of n up to 100. This collection contains a wide variety of matrices (for example, real, complex, well-conditioned, ill-conditioned, structured, contrived, practically occurring), and many of them are parametrized. Our experiments led to a new discovery, which is described in the next section.

2. Number of iterations. As mentioned above, Algorithm 1 usually requires at most four iterations and it never requires more than $n+1$. In our numerical experiments we found one particular family of matrices from [10] for which, depending on n, up to n iterations were required (the matrices are $B = \mathrm{inv}(\mathrm{fiedler}(\mathrm{seqm}(-1,2,\mathrm{n})))$ in the notation of [10]). Consideration of this example led us to construct a symmetric $n \times n$ tridiagonal matrix $T_n(\alpha) = (t_{ij})$ for which it can be proved that n iterations are required by Algorithm 1 if $0 \le \alpha < 1$. The matrix is defined by

$$\begin{aligned} t_{ii} &= \begin{cases} 2, & i = 1, \\ i, & 2 \le i \le n-1, \\ -t_{n,n-1} + \alpha, & i = n, \end{cases} \\ t_{i,i+1} &= \begin{cases} -((i+1)/2 - \alpha) & \text{if } i \text{ is odd}, \\ -i/2 & \text{if } i \text{ is even}. \end{cases} \end{aligned}$$

To illustrate,

$$T_6(\alpha) = \begin{bmatrix} 2 & -(1-\alpha) & & & & \\ -(1-\alpha) & 2 & -1 & & & \\ & -1 & 3 & -(2-\alpha) & & \\ & & -(2-\alpha) & 4 & -2 & \\ & & & -2 & 5 & -(3-\alpha) \\ & & & & -(3-\alpha) & 3 \end{bmatrix}.$$

Note that, for all α, an optimal x for Algorithm 1 is $x = e_{n-1}$, that is, $\|T_n(\alpha)e_{n-1}\|_1 = \|T_n(\alpha)\|_1$. It is straightforward to show that if Algorithm 1 is applied to $T_n(\alpha)$ with $0 \le \alpha < 1$ then $x = e_{i-1}$ on the ith iteration, for $i = 2, \cdots, n$, with convergence on the nth iteration. The same would be true of Algorithm EST if it were not limited to five iterations. For $n \ge 5$, after five iterations we have $y^5 = T_n(\alpha)e_4$, so $\|y^5\|_1 = 8-\alpha$. Since $\|T_n(\alpha)\|_1 = 2n - 2 - \alpha$, we have

$$\frac{\|y^5\|_1}{\|T_n(\alpha)\|_1} = \frac{8-\alpha}{2n-2-\alpha} \to 0 \quad \text{as } n \to \infty.$$

For $\alpha \ge 1$ we obtain convergence to $x = e_1$ on the second iteration and, similarly, the underestimation ratio $\|y^2\|_1/\|T_n(1)\|_1 \to 0$ as $n \to \infty$. For $\alpha < -1$ Algorithm 1 requires $n-1$ iterations, and $x = e_i$ on the ith iteration. Fortunately, in all cases the "extra vector" v in (3) enables Algorithm EST to produce a good norm estimate. For $\alpha \le 1$ we have

$$T_n(\alpha)v = |T_n(\alpha)||v| \ge |T_n(\alpha)|e,$$

which implies

$$\|T_n(\alpha)v\|_1 \geq \| \, |T_n(\alpha)|e \, \|_1 = n^2 - n\alpha,$$

and hence

$$\frac{\|T_n(\alpha)v\|_1/\|v\|_1}{\|T_n(\alpha)\|_1} \geq \frac{(n^2 - n\alpha)/(\frac{3}{2}n)}{2n - 2 - \alpha} \geq \frac{1}{3}.$$

For $\alpha > 1$ it does not seem possible to obtain a concise lower bound for $\|T_n(\alpha)v\|_1$, but a rough analysis indicates that the extra estimate cannot differ greatly from $\|T_n(\alpha)\|_1$ (and this is supported by numerical tests). In conclusion, Algorithm EST produces an acceptable estimate of $\|T_n(\alpha)\|_1$ for all n and α. We are not aware of any matrices for which the limit of five iterations in Algorithm EST is responsible for the algorithm producing a poor estimate.

3. Worst-case norm estimates. A class of matrices $B(\theta)$ is given in [9] for which $\gamma/\|B(\theta)\|_1$ can be arbitrarily small for Algorithm EST. In practice, rounding errors sustained in constructing $B(\theta)$ usually make the computed matrix one for which a good estimate is returned. Apart from the matrices $B(\theta)$ we have found exceedingly few matrices for which the estimate γ is less than $\frac{1}{3}\|B\|_1$. Here we list a few of the worst cases. Where $A = B^{-1}$ is specified we used Algorithm EST to estimate $\|A^{-1}\|_1$, which involves solving linear systems with A and A^T as coefficient matrices. $QR(B)$ denotes the triangular QR factor of B. The matrices are specified using the notation of [10]; they are of orders 16, 8, 50, and 100, respectively.

$A = \text{vand}(\text{seqa}(-1, 1, 16))$,	$\gamma/\|B\|_1 = 0.199$,	
$A = \text{vand}(\text{seqa}(-1, 1, 8))$,	$\gamma/\|B\|_1 = 0.288$,	
$B = QR(\text{chebspec}(50))$,	$\gamma/\|B\|_1 = 0.326$,	
$A = \text{vand}(100)$,	$\gamma/\|B\|_1 = 0.207$,	$\|B\|_1 = 6.66 \times 10^{49}$.

In the last example Algorithm EST detected cycling. This matrix A is so violently ill-conditioned that the computed quantities in Algorithm EST may have no correct significant digits (the same applies to the "exact" $\|A^{-1}\|_1$ with which we compare γ!). These results were obtained using our PC-MATLAB implementation of Algorithm EST. PC-MATLAB uses IEEE standard arithmetic with machine epsilon $2^{-52} \approx 10^{-16}$. Different results might be obtained in another computing environment. To indicate the sensitivity of these worst-case estimates to rounding errors we mention that when A in the last example was scaled to $A/3$ Algorithm EST produced an exact estimate!

4. Applications. The most obvious application of Algorithm EST is to the estimation of the norm condition number $\kappa_p(A) = \|A\|_p\|A^{-1}\|_p$, $p = 1, \infty$. We summarize three other applications in which the algorithm has been found to be useful.

(1) In [12] a hybrid algorithm is developed for computing the polar decomposition. Algorithm EST is used to decide when to switch from one iteration to another by testing the convergence criterion $\|X_k^T X_k - I\|_1 < 1$, where $X_k \in \mathbb{R}^{n\times n}$. By using Algorithm EST formation of the matrix product $X_k^T X_k$ is avoided.

(2) When a square linear system $Ax = b$ is subject to perturbations ΔA and Δb satisfying inequalities $|\Delta A| \leq \epsilon E$, $|\Delta b| \leq \epsilon f$, a bound for the change in x can be derived that involves the condition number

$$\kappa_{E,f}(A, b) = \frac{\| \, |A^{-1}|E|x| + |A^{-1}|f \, \|_\infty}{\|x\|_\infty}.$$

The following idea from [1] shows how $\kappa_{E,f}(A,b)$ can be estimated with the aid of Algorithm EST. Defining $z = (E|x| + f)/\|x\|_\infty$ and $Z = \mathrm{diag}(z)$ we have

$$\begin{aligned}\kappa_{E,f}(A,b) &= \|\, |A^{-1}|z \,\|_\infty = \|\, |A^{-1}|Ze \,\|_\infty \\ &= \|\, |A^{-1}Z|e \,\|_\infty = \|\, |A^{-1}Z| \,\|_\infty \\ &= \|\, A^{-1}Z \,\|_\infty = \|B\|_1,\end{aligned}$$

where $B = (A^{-1}Z)^T$. To apply Algorithm EST we just need to evaluate products $Bx = Z(A^{-T}x)$ and $B^T\xi = A^{-1}(Z\xi)$, which is easily done given the ability to solve linear systems involving A and A^T. LAPACK [3] uses Algorithm EST to provide an estimate of $\kappa_{E,f}(A,b)$ in routines for the solution of $Ax = b$ by Gaussian elimination with iterative refinement.

(3) The componentwise perturbation analysis mentioned in (2) can be extended to least squares problems [2], [4], [11] (i.e., to rectangular A and E), and a bound is obtained that contains the terms

$$\frac{\|\, |A^+|(E|x| + f) \,\|_\infty}{\|x\|_\infty} \quad \text{and} \quad \frac{\|\, |(A^TA)^{-1}|E^T|r| \,\|_\infty}{\|x\|_\infty},$$

where A^+ is the pseudo-inverse of A and $r = b - Ax$. As in (2), each of these terms can be reduced to the form $\|B\|_1$, where it is straightforward in the context of solving a least squares problem to evaluate Bx and $B^T\xi$.

Acknowledgments. I thank Des Higham for pointing out an oversight in my original analysis of the $T_n(\alpha)$ example. The associate editor Robert Skeel also offered useful comments.

REFERENCES

[1] M. Arioli, J.W. Demmel, and I.S. Duff, *Solving sparse linear systems with sparse backward error*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 165–190.

[2] M. Arioli, I.S. Duff, and P.P.M. de Rijk, *On the augmented system approach to sparse least-squares problems*, Numer. Math., 55 (1989), pp. 667–684.

[3] C.H. Bischof, J.W. Demmel, J.J. Dongarra, J.J. Du Croz, A. Greenbaum, S.J. Hammarling, and D.C. Sorensen, *Provisional contents*, LAPACK Working Note #5, Report ANL-88-38, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 1988.

[4] A. Björck, *Componentwise backward errors and condition estimates for linear least squares problems*, Manuscript, Department of Mathematics, Linköping University, Sweden, March 1988.

[5] R.W. Brankin and I. Gladwell, *Codes for almost block diagonal systems*, Internat. J. Computers and Mathematics with Applications, 19 (1990), pp. 1–6.

[6] T.F. Coleman and L.A. Hulbert, *A direct active set algorithm for large sparse quadratic programs with simple bounds*, Tech. Report TR 88-926, Department of Computer Science, Cornell University, Ithaca, NY, 1988.

[7] W.W. Hager, *Condition estimates*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 311–316.

[8] N.J. Higham, *A survey of condition number estimation for triangular matrices*, SIAM Rev., 29 (1987), pp. 575–596.

[9] ———, FORTRAN *codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation* (*Algorithm 674*), ACM Trans. Math. Software, 14 (1988), pp. 381–396.

[10] ———, *A collection of test matrices in* MATLAB, Tech. Report 89-1025, Department of Computer Science, Cornell University, Ithaca, NY, 1989.

[11] ———, *Computing error bounds for regression problems*, in Proc. Joint Summer Research Conference on Statistical Analysis of Measurement Error Models and Applications, P.

Brown and W.A. Fuller, eds., Contemporary Mathematics, American Mathematical Society, Providence, RI, 1990.

[12] N.J. Higham and R.S. Schreiber, *Fast polar decomposition of an arbitrary matrix*, Tech. Report 88-942, Department of Computer Science, Cornell University, Ithaca, NY, 1988; SIAM J. Sci. Statist. Comput., this issue, pp. 648–655.

[13] NAG, NAG *Fortran Library Manual*, Mark 13, NAG Ltd., Oxford, U.K., 1988.

[14] Pham Dinh Tao, *Convergence of a subgradient method for computing the bound norm of matrices*, Linear Algebra Appl., 62 (1984), pp. 163–182. (In French.)

[15] ———, *Some methods for computing the maximum of quadratic form on the unit ball of the maximum norm*, Numer. Math., 45 (1984), pp. 377–401. (In French.)

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 5, pp. 811-823, September 1990

A BLOCK ORDERING METHOD FOR SPARSE MATRICES[*]

JAMES O'NEIL[†] AND DANIEL B. SZYLD[‡]

Abstract. Block iterative methods used for the solution of linear systems of algebraic equations can perform better when the diagonal blocks of the corresponding matrix are carefully chosen. A method is presented based on combinatorial considerations which permutes the rows and columns of a general matrix in such a way that relatively dense blocks of various sizes appear along the diagonal. The method is particularly useful when no natural partitioning of the matrix is available. Two parameters govern the method which is $O(n+\nu)$ in time and space, where n is the order of the matrix and ν is the number of nonzeros in the matrix. Numerical test results are presented which illustrate the performance of both the ordering algorithm and the block iterative methods with the resulting orderings.

Key words. sparse matrices, block methods, partition, ordering

AMS(MOS) subject classifications. 65F50, 65F10, 68P10, 05C30, 05C50

1. Introduction.

Efficient methods for the solution of large sparse linear algebraic systems of equations of the form

$$Ax = b, \tag{1}$$

where A is an $n \times n$ real general matrix and x and b are n-vectors, have been studied and successfully implemented during the last few decades. High-quality software now exists for the solution of (1) using different direct or iterative methods. For example, ITPACK [11] contains programs for several classical iterative methods, including Jacobi, Gauss–Seidel, successive overrelaxation (SOR) and others; see Hageman and Young [10], Varga [19], or Young [20] for their descriptions. For direct methods, SPARSPAK [8] is usually recommended for the symmetric case and MA28 [3] for the nonsymmetric case. In addition, the recent package NSPCG [13] contains many nonsymmetric preconditioned conjugate gradient procedures.

Classical block iterative methods, which we briefly review in the next section, are attractive, since they combine the properties of (point-) iterative methods for very large systems (such as those arising from the discretization of three-dimensional differential equations or from massive electronic circuit simulations) with those of direct methods for the solution of smaller systems in each of the diagonal blocks. In fact, if the diagonal blocks in question are very dense, codes for full factorization are used, and no overhead, as is typical of sparse matrix codes, is incurred. In addition, using a block method can, in many cases, reduce the spectral radius of the iteration matrix [19, Thm. 3.15], and, depending upon the method of solution of the diagonal blocks and the machine architecture, the overall convergence time can be greatly reduced.

The question remains of how to best partition the matrix A into blocks. The block structure of a matrix depends upon its zero-nonzero structure and not upon the values of its entries. In other words, the partitioning of the set $I_n = \{1, \cdots, n\}$,

[*] Received by the editors April 16, 1989; accepted for publication November 2, 1989.

[†] BDM International, Inc., 7915 Jones Branch Drive, McLean, Virginia 22102 (oneil@bdmrrr.bdm.com).

[‡] Department of Computer Science, Duke University, Durham, North Carolina 27706 (na.szyld@na-net.stanford.edu). Present address, Department of Mathematics, Temple University, Philadelphia, Pennsylvania 19122. The work of this author was supported by National Science Foundation grant DMS-8807338.

which determines the block structure, corresponds to the grouping of nodes in the graph of the matrix [4], [8], [15], [17]. In cases where the matrix corresponds to the discretization of a differential equation on a regular domain, a regular partition of the domain provides a corresponding partition of the matrix [19, Chap. 6], [14]. Similarly, when large finite element structures are partitioned into substructures, each substructure may correspond to a block of the matrix. If the domain is irregular, or the matrix does not correspond to a differential equation, no systematic way of partitioning has been available. In this paper we remedy that situation; we present a method that partitions I_n in such a way that relatively dense blocks of various sizes appear along the diagonal. The algorithm presented, called PABLO (PArameterized BLock Ordering), is linear in time and space and thus contributes very little overhead to the overall computation. If a natural partition of the graph of the matrix exists, the partition produced by PABLO is still competitive in the sense that if the same block iterative method is used with both partitions, the numbers of operations required for convergence are quite similar. In §5, we present some results illustrating this.

Other algorithms have been devised which permute and partition a matrix in order to improve the behavior of a sparse direct factorization; see, for example, the nested dissection algorithm by George [7], [8], [15]; and the recent paper by Pothen, Simon, and Liou [16]. To our knowledge, our work represents the first time a partitioning algorithm has been specially designed for block iterative methods.

In §2.2 we review the notation and concepts of graph theory needed for the description of PABLO. The crux of our method, described in detail in §3, is to choose groups of nodes in G, the graph of the matrix, so that the corresponding diagonal blocks are either full or very dense. The choices are made by starting a group with a given node in G and adding a new node to that group if either of two criteria are satisfied. The first criterion is that the new group must be in some sense as full as the group without the new node. This is accomplished by computing the ratio of the number of edges to the total number of possible edges in the resulting subgraph. This ratio must be no less than a specified parameter, α, multiplied by the corresponding ratio for the graph without the new node. The second criterion is that the new node must be adjacent to more nodes in the group than outside the group.

In this paper we introduce PABLO and show that it is an effective tool for partitioning matrices into blocks. This is a preprocessing step before the use of block iterative methods, and since it is successful even when a natural partition exists, it can be used as a "black box" for all cases.

2. Preliminaries.

2.1. Block iterative methods.

Let us consider the partitioning of

$$I_n = \{1, \cdots, n\}$$

into q mutually disjoint subsets $J_k = \{p_1^k, \cdots, p_{s_k}^k\} \subseteq I_n$, $k = 1, \cdots, q$. Of course, $\sum_{k=1}^q s_k = n$. Consider the permutation π of I_n, where $\pi(1) = p_1^1$, $\pi(2) = p_2^1, \cdots, \pi(s_1) = p_{s_1}^1$, $\pi(s_1 + 1) = p_1^2, \cdots, \pi(s_1 + s_2) = p_{s_2}^2, \cdots, \pi(n) = p_{s_q}^q$. Let us call P the permutation matrix corresponding to π [2], [8], [19]. Then the system (1) is equivalent to

$$(PAP^T)Px = Pb. \tag{2}$$

The matrix and vectors in (2) can be partitioned according to the sets J_k as pictured below, where the matrix PAP^T is partitioned into a $q \times q$ array of smaller matrices, and Px and Pb are partitioned into vectors composed of q subvectors.

$$\begin{bmatrix} A_{11} & A_{12} & \dots & A_{1q} \\ A_{21} & A_{22} & \dots & A_{2q} \\ \vdots & \vdots & \vdots & \vdots \\ A_{q1} & A_{q2} & \dots & A_{qq} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_q \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_q \end{bmatrix}. \tag{3}$$

Block iterative methods such as block Gauss–Seidel, block Jacobi and block SOR are generalizations of the corresponding point-iterative methods [19], [20]. Let x_k^r denote the kth subvector, of dimension s_k corresponding to J_k, of the vector Px at the rth iteration. The Gauss–Seidel method, as applied to (3) and described by the following set of equations, is representative of the methods to be considered.

For $k = 1, \cdots, q$

$$y_k^r = \left(b_k - \sum_{s=1}^{k-1} A_{ks} x_s^r - \sum_{s=k+1}^{q} A_{ks} x_s^{r-1} \right) \tag{4}$$

$$A_{kk} x_k^r = y_k^r, \tag{5}$$

where it is assumed that the $s_k \times s_k$ diagonal blocks, A_{kk}, are nonsingular. The study of convergence for this and similar block methods is analogous to that of point methods and can be found in [10], [19], [20].

2.2. Graph theory for matrix structures. The use of graph theory for the analysis of matrix structures had been applied primarily to irreducible matrices (see [2], [19], for example); however, during the last two decades, its use has been extended to the analysis of Gaussian elimination and sparse matrices in general [3], [8], [15], [17]. In this section we review some concepts of graph theory associated with matrix structures, present our notation, and introduce new definitions needed to describe the formulation of our block partitioning method.

A *graph* G is defined as an ordered pair (V, E), where V is a set of vertices and E is a set of pairs of elements of V that represent edges in the graph. The graph $G(A)$ induced by a square matrix $A = [a_{ij}]$, of order n, has $|V| = n$, where a vertex i, in V, corresponds to row (or column) i in the matrix A. If a matrix A is nonsingular, it can be permuted in such a way that $a_{ii} \neq 0$ [6], [9]. In our method, the matrix A is first permuted so that its diagonal is zero free, and, without loss of generality, we assume that $a_{ii} \neq 0$ for all i. If A cannot be permuted so that $a_{ii} \neq 0$ for all i, then the matrix is symbolically singular, and that fact will be discovered during the mentioned step. An edge $(i, j) \in E$ if and only if $a_{ij} \neq 0$ and $i \neq j$ (given the assumption above that $a_{ii} \neq 0$). For nonsymmetric matrices the edges are directed, and we consider the *ordered* pair (i, j), while in the symmetric case, the edges are undirected and $(i, j) \in E \Leftrightarrow (j, i) \in E$.

A vertex v is said to be *adjacent* to a vertex w if $(v, w) \in E$ or $(w, v) \in E$. The *degree* of a vertex v, $deg(v)$, is the number of vertices adjacent to v. A *clique* is a graph where $(i, j) \in E \ \forall i \neq j, \ i, j \in V$. In the case of a matrix-induced graph $G(A)$, a clique corresponds to a full matrix, that is, to a matrix where $a_{ij} \neq 0 \, \forall i, j \in I_n$.

Given $S \subseteq V$, we call $G(S) = (S, F)$ a *section subgraph* of (V, E) in which the set of edges, F, consists of exactly those edges in E with both endpoints in S [8, p. 38]. In the case of a matrix-induced graph $G(A)$, the section subgraph $G(S)$ is the graph induced by some submatrix or block. We define the *fullness* of $G(S)$, ϕ_S, as the number of edges in F divided by the number of edges in a clique with the same number of vertices as in S. Thus, $\phi_S = 2\,|F|\,/(|S|^2 - |S|)$ for undirected graphs, and $\phi_S = |F|\,/(|S|^2 - |S|)$ for directed graphs. Note, $0 \le \phi_S \le 1$ for any $G(S)$, and $\phi_S = 1$ if and only if $G(S)$ is a clique.

Consider a graph $G = (V, E)$ partitioned into q section subgraphs $G(W_k) = (W_k, E_k)$, where $\bigcup_{k=1}^q W_k = V$ and $W_r \cap W_s = \emptyset$ for $r \neq s$. The sets of nodes W_k correspond to the sets J_k and to the permutation π, which determines them as described in §2.1. Note that $\bigcup_{k=1}^q E_k \subseteq E$ and that equality does not hold in general. If equality holds, the permutation π has reduced the matrix to block diagonal form, where $A_{rs} = 0$, $r \neq s$ in (3).

A *path* (of length k) joining two nodes i, $j \in V$ is a set of k "consecutive" edges (i, i_1), (i_1, i_2), $\cdots$, (i_{k-1}, j). A subgraph $G(S) = (S, F)$ is said to be *strongly connected* if for every pair i, $j \in S$ there is a path from i to j formed by elements in F. The decomposition of a graph into its *strongly connected components* is unique. Efficient algorithms to determine this decomposition are available (see [6], [9], [18]). Given the strongly connected components of the graph one can build the corresponding sets W_k and the permutation π by which A can be transformed to lower (upper) block triangular form, i.e., $A_{rs} = 0$, $r > s$ $(r < s)$ in (3). Once this is done, the linear system (3) is decoupled into q smaller linear systems.

3. PABLO algorithm.

The PABLO algorithm partitions a graph $G = (V, E)$ into q subgraphs $G(V_k) = (V_k, E_k)$, where $\bigcup_{k=1}^q V_k = V$ as described in §§2.1 and 2.2. The goal is to establish easily computable criteria to define the sets V_k in such a way that $G(V_k)$ is tightly connected by some measure, which would imply that the diagonal blocks are relatively full. This in turn will, in some cases, result in faster convergence rates for block iterative methods [19, Thm. 3.15]; see also §5. In our method, the matrix is first permuted into its block triangular form, and the core of the algorithm is applied to each nontrivial diagonal block. Thus, we assume in our description that the matrix is irreducible, that is, that its graph is strongly connected.

We describe the algorithm constructively, selecting the nodes for each of the sets V_k, $k = 1, \cdots, q$. Note that q, the number of blocks, is not known a priori, but is instead computed by PABLO as the number of groups of nodes obtained at completion. Let V be the set of vertices of the (strongly connected) graph. Furthermore, consider three sets P, Q, and C that form initially a partition of V. We may then view the block partitioning algorithm, PABLO, as a series of operations that manipulate the elements of the sets of the partition until no other operations can be applied. Alternatively, PABLO can be viewed as a search [1] to mark the nodes of the graph in a specific order. The set P contains those vertices that will be assigned to the current block and, at the end of the step, becomes the current V_k. The set Q, contains vertices (in the current connected component) which are adjacent to at least one vertex in P and are thus considered "eligible" to be added to the block. The set Q is maintained as a queue so that nodes closer to the nodes already added to P will be investigated first. The set C consists of those unassigned vertices in V neither in P nor in Q. The use of the set Q allows us to limit the search for new additions to P to a set of "eligible" nodes. It is the use of this set which allows the algorithm to have a linear time complexity.

Consider the vertices that are not yet part of a completed block. For the vertex i, let $deg(i)$ be the number of edges of the form (i, j) or (j, i), for some j, and let $in(i)$, at every point in the algorithm, be the number of edges of the form (i, j) or (j, i) where $j \in P$. Clearly, $in(i) \le deg(i)$ at all times. The definitions of in and deg imply

```
     Produce a representation for $G$, the graph induced by the matrix
     Initialize all nodes as unmarked
     Find the (strongly) connected components of the graph
     Place all components of one node in block, mark these nodes
     **For** each remaining connected component **do**
         let $C = \{vertices\ in\ connected\ component\}$
$\ell1$:     **repeat**
             let $P = \emptyset$, and $Q = \emptyset$
             choose from $C$ a node $c$, mark it, and place it in $P$
             move to $Q$ all nodes in $C$ adjacent to $c$
             update $in$ for all nodes in $Q$
             set $\phi_P = 0$
$\ell2$:         **repeat**
                 choose the node $p$ from the head of $Q$
                 calculate $\phi_{P\cup\{p\}}$
                 **if** ($\phi_{P\cup\{p\}} \geq \alpha\phi_P$) **or** ($in(p)/deg(p) \geq \beta$) **then**
                     mark $p$ and move $p$ to $P$
                     add to the rear of $Q$ all nodes in $C$ adjacent to $p$
                     update $in$ for all vertices in $Q$ that are adjacent to $p$
                     update $\phi_P$
                 **else**
                     move node $p$ from $Q$ to $C$
                 **endif**
             **until** $Q = \emptyset$
             designate those nodes in $P$ to be in a block
$\ell3$:         **for** all nodes $c$ in $C$ **do**
                 update $deg(c)$ to reflect the marking of the nodes added to $P$
         **until** $C = \emptyset$
     **endfor**
```

FIG. 1. PABLO: *Block ordering algorithm.*

that the symmetric permutation for a nonsymmetric matrix, A, which is produced by PABLO will differ from that produced for the symmetric matrix $(|A| + |A^T|)$, where $|A|$ is the matrix with entries $|a_{ij}|$ and A^T is the transpose of A. To start the method, we set $P = Q = \emptyset$ and $C = V$. We choose some node c from C, mark it, and add it to the set P. We set ϕ_P, the fullness of $G(P)$, to zero (recall we do not include the self-edges). We then move to the rear of Q all those vertices in C that are adjacent to the vertex just added to P and increment in for those vertices. Next, we take a node p, from the front of Q, calculate $\phi_{P\cup\{p\}}$, the fullness of $G(P \cup \{p\})$, and add p to P if either of the following two criteria is satisfied.

(1) The fullness of $G(P \cup \{p\})$ is at least some fraction of the fullness of $G(P)$; that is, $\phi_{P\cup\{p\}} \geq \alpha\phi_P$, where $0 \leq \alpha$.

(2) The node p is adjacent to more nodes in P than in $Q \cup C$; that is, $in(p)/deg(p) \geq \beta$, where $0 \leq \beta \leq 1$.

If we exhaust the set Q, we define the current $V_k = P$. If the set C is also empty, then we have numbered all the nodes in the (strongly connected) graph. If, however, Q is empty but C is not, then there is no vertex satisfying the criteria above, and for all vertices c in C that are adjacent to vertices in V_k, we update $deg(c)$. This operation is done to reflect the fact that nodes in V_k no longer have an effect on the unmarked vertices in C. For all unmarked nodes, we set in to 0 and then set $P = \emptyset$ and repeat the method (with $k \leftarrow k + 1$) as if C represented a new graph.

The complete algorithm is given in Fig. 1.

The existence of the two parameters α and β allows the user to tailor the block-partitioning and thereby the solution of the system of equations to particular time and space requirements. The choice for α gives the user some control over how densely connected the blocks will be. The value for α can vary from zero to infinity. If $\alpha = 0$, then the first criterion is essentially disabled, for the addition of any vertex into P will satisfy $\phi_P \geq 0$. When $\alpha > 1$, the fullness of P must increase. If the graph is undirected, the first criterion cannot be satisfied once the second node is added, unless vertices are added via the second criterion. Thus, a value greater than one is very restrictive and will result in small blocks. Empirically, the values of α producing the best block partitionings lie between 0.7 and 1.2. Larger values of α yield smaller, but more tightly connected, blocks. See [12] for results supporting this range of values for α.

By varying the value of β, the user can admit to P those vertices that are tightly connected to vertices within P but whose inclusions into P do not preserve the fullness of $G(P)$. A value of $\beta = 0$ allows any vertex to be added to the block. This will of course yield one block for each (strongly) connected component. A value of $\beta = 1.0$ would allow a vertex q to be admitted to P only if all the edges with which q is incident have their other endpoints in P, that is, if $in(q) = deg(q)$. This is similar to the first criterion with $\alpha = 1.0$; however, if P is tightly connected, and $|P| \gg deg(q)$, then even if $in(q) = deg(q)$, the addition of q may lower ϕ_P, and the first criterion would not be met, despite the fact that the addition of q to P is desirable. Since all of the vertices adjacent to q are in P, it is clear that q should be added to the current block; the second criterion allows this to occur. A natural choice for β is 0.5; that is to say, add a node q to P if there are more edges between q and vertices in P than edges between q and vertices not in P. In order to admit a node q to P, a value greater than 0.5 would require that the number of edges directed into P from q be greater than the number of edges not directed into P from q, whereas a value of less than 0.5 would allow vertices with more edges to nodes outside of P than to nodes in P to be added to P. As one might expect, larger values of β produce a larger number of small, but tightly connected, blocks, whereas smaller values yield a smaller number of large, but less densely connected, blocks.

4. Complexity. We present an analysis of the algorithm PABLO as described in Fig. 1. We first show that the algorithm PABLO will always terminate and then show the complexity of the two main loops in PABLO, namely, $\ell 1$ and $\ell 2$ (see Fig. 1). Table 1 summarizes the complexity analysis of the algorithm, where γ_i is the number of edges in the connected component Γ_i, n is the number of nodes, and ν is the total number of edges, i.e., the number of nonzeros in the matrix. The following analysis shows that the complexity of the algorithm is proportional to ν and n. In §5 this linearity is illustrated with a series of experiments. Note that the linearity would be lost if the search for nodes to be added to P were unrestricted and not just limited to Q. It is easy to verify that PABLO's space requirements are also proportional to ν and n.

LEMMA 4.1. *The algorithm* PABLO *always terminates.*

Proof. We first point out that the number of nodes, n, and the number of edges, ν, are finite, and there are at most n connected components. At any given time, the size of Q is bounded. The loop $\ell 2$ (see Fig. 1) terminates, since the value of the ordered pair (number of unmarked nodes, size of Q) is lexicographically smaller at the end of each iteration than it was at the end of the previous iteration. The loop $\ell 1$ terminates because the size of C decreases by a least one in every iteration (cf. line 9 of the algorithm in Fig. 1). □

TABLE 1
PABLO *complexity analysis.*

Operation	Timing
Initialization	$O(n+\nu)$
Finding (strongly) connected components	$O(n+\nu)$ ([18] or [1, p. 176])
Finding components with one node	$O(no.\ of\ components) \Rightarrow O(n)$
Updating ϕ_P	$O(1) \Rightarrow O(n)$
Marking nodes of new P	$O(n)$
Updating degrees of nodes in C not added to P	$O(\gamma_i)$ for each $\Gamma_i \Rightarrow O(\nu)$
PABLO Timing Complexity	$O(n+\nu)$

Since we have shown that the method terminates, we turn now to the analysis of the complexity of its execution. The initialization steps in lines 1 and 2 (in Fig. 1) can be performed in $O(n+\nu)$ and $O(n)$ time, respectively, while finding the strong components of a graph takes also the same order of operations [18], [1, p. 176]. The heart of the algorithm is the loop $\ell 2$.

LEMMA 4.2. *The complexity of the loop $\ell 2$ in the algorithm* PABLO *is $O(\nu)$, i.e., it is proportional to the number of nonzeros in the matrix.*

Proof. Let γ_i be the number of edges in the component $G(\Gamma_i)$. In $\ell 2$, nodes are moved from Q to P and between Q and C. A vertex q can be added to Q at most $deg(q)$ times. A vertex q is added to Q only when some vertex p has just been added to P, and there is an edge between p and q. Since a node that has been added to P is marked and cannot be added to P in a subsequent step, it follows that if q is subsequently removed and then added again to Q, it is only because of its adjacency with some *other* vertex, $\tilde{p}$. Therefore, there can be at most $\sum_{q\in\Gamma_i} deg(q) = 2\gamma_i$ additions to Q. Since on each iteration of $\ell 2$ a node is moved from Q to either P or C, there are then at most $2\gamma_i$ additions to C and P. Since C must be empty at the termination of $\ell 1$, it follows that all the nodes in Γ_i will have been added to a P (perhaps not the same P, however) in $O(\gamma_i)$ time. This is done for all the (strongly) connected components, and since $\sum_i \gamma_i \leq \nu$, the number of nonzeros in the entire matrix, the number of operations for this step is $O(\nu)$. The updates of the value of *in* are bounded by the number of edges in the entire graph, namely, ν, since only those adjacent to p, which is only marked once, have to be processed at the step in line 19. Thus, the loop $\ell 2$ takes $O(\nu)$ operations. □

Similar analyses for the updates of *in* and *deg* in lines 11 and 27 yield a complexity of $O(\nu)$ for the loop $\ell 1$. The overall complexity of the PABLO algorithm is then $O(n+\nu)$.

5. Experimental results. In this section we illustrate with sample experimental results, obtained on a Convex C1 computer at Duke University, that PABLO is an effective algorithm as a preprocessor of linear algebraic systems of equations before the application of a block iterative method. In all experiments reported here, the values of $\alpha = 0.5$ and $\beta = 1.0$ have been used. The point we wish to stress is that the main application envisioned for PABLO is in the cases where no good partitioning of the matrix is known to the user. For example, consider the finite element discretization of order 3025 of a graded L-shaped region [8, Chap. 9] with -1 in the off-diagonal positions and $\sum_{j\neq i} |a_{ij}| + .01$ in the ith diagonal position. This represents a discretization of a Dirichlet problem. Table 2 presents statistics on the solution of that linear system using different methods. The number of blocks indicates the number of diagonal block systems that must be factored and then solved for each block iteration. The table also presents the number of operations required to factor all of the diagonal blocks. The number of iterations to reach convergence and the number of operations per iteration are also provided. The total number of operations is then the product of the number of iterations times the number of operations per iteration, plus the operations needed to factor the diagonal blocks. Both in the case of Gauss–Seidel and SOR, the block method with the ordering provided by the PABLO algorithm requires, as expected, fewer operations than the point method. Results presented for each SOR method are those with the value of ω for which convergence was attained with the fewest number of operations.

TABLE 2

Operation count for different methods. L-shaped grid.

Graded L-shaped	Number of blocks	Factor diag. blocks (ops.)	Number of iter.	Solve (ops./iter.)	Total (ops.)
Point Gauss-Seidel			5,603	44,691	250,403,673
PABLO Gauss-Seidel	640	27,363	3,040	47,027	142,989,443
Point SOR, $\omega = 1.91$			229	53,766	12,312,414
PABLO SOR, $\omega = 1.87$	640	27,363	141	56,102	7,937,745

We also performed a series of experiments on the 9-point discretization of the Laplacian operator on a square grid. In this case, we can compare the performance of the partition produced by PABLO with those of the natural partitions of the grid. The 10×10 grid is shown in Fig. 2 together with dashed lines grouping the nodes chosen by the algorithm PABLO. The rows and columns corresponding to these nodes form the diagonal blocks to be factored in a block iterative method. Table 3 presents the same kind of statistics as in Table 2 for different SOR methods in the case of a 30×30 grid, a matrix of order 900. We point out that the block iterative method using the partitioning performed by PABLO takes fewer operations than if the blocks corresponding to lines of the grid are used. The results with other partitions reported in Table 3 correspond to squares of 2, 3, 5, 10, and 15 nodes per side. It can be seen that for some block partitions, the corresponding iterative method required slightly fewer operations than the one corresponding to the partition produced by PABLO. In other words, the performance of the partition produced by PABLO is close to the best observed. Again, results presented for each SOR method are those with the value of ω for which convergence was attained with the fewest number of operations.

TABLE 3

Operation count for different methods. 30×30 *grid.*

30×30 grid SOR method	ω	Number of blocks	Factor diag. blocks (ops.)	Number of iter.	Solve (ops./iter.)	Total (ops.)
Point	1.80			94	19,088	1,794,272
Line	1.76	30	4,380	74	19,208	1,425,772
"2×2 squares"	1.75	225	8,325	62	19,088	1,268,133
"3×3 squares"	1.71	100	16,900	55	19,888	1,110,740
"5×5 squares"	1.64	36	51,012	45	24,848	1,169,172
"10×10 squares"	1.54	9	174,960	32	37,880	1,387,120
"15×15 squares"	1.50	4	368,804	28	50,352	1,778,660
PABLO	1.75	154	9,627	66	19,688	1,309,035

Figure 3, obtained with data from different grid sizes of the 9-point discretization of the Laplacian, illustrates the linearity of the PABLO algorithm, confirming the analysis in §4. We point out that in PABLO, no floating point operations are performed, only assignments and conditional statements are needed. Thus the "operation count" in this graph refers to operations that take about 10 times less time than a floating point operation.

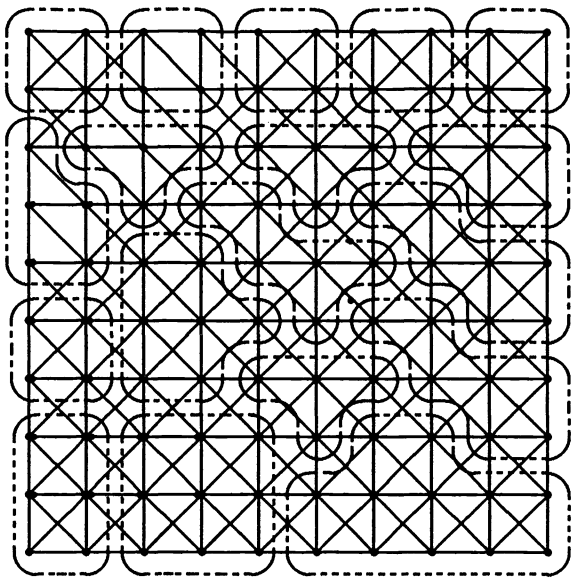


FIG. 2. *Groups of nodes chosen by* PABLO *in a* 10×10 *grid.*

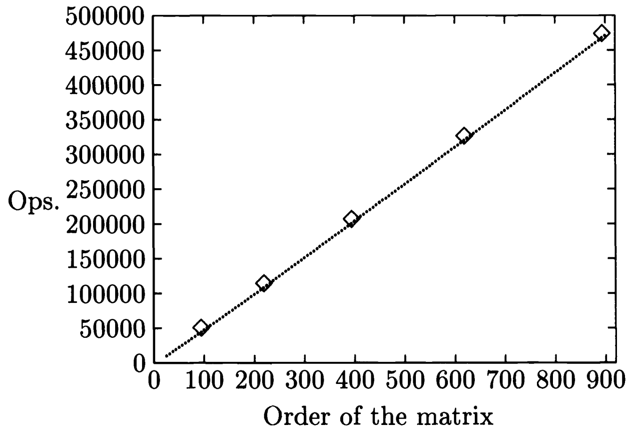


FIG. 3. *Assignments and conditional statements in* PABLO.

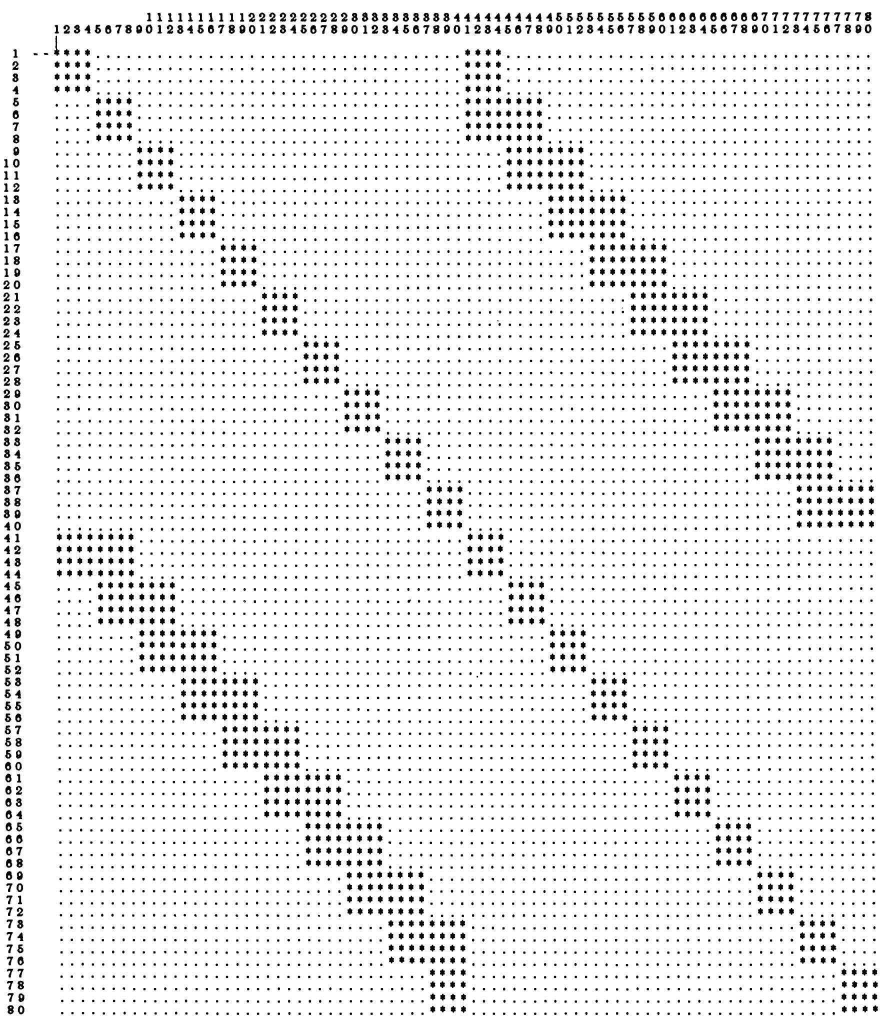


FIG. 4. *Zero-nonzero structure of "steam" in its original order.*

Results similar to those in Tables 2 and 3 were observed with matrices from other applications. For example, in the case of circuit simulation, one can partition the circuit into certain groups of devices. This partitioning can be performed by the user when the size of the circuit is manageable, say with 100 components. In this case, since the user has knowledge of the values in the matrix, the choice is more informed and a block iterative method converges faster than the same method using the partition generated by PABLO, which takes into account only the location of the nonzeros. Nevertheless, both execution times are of the same order of magnitude. When the

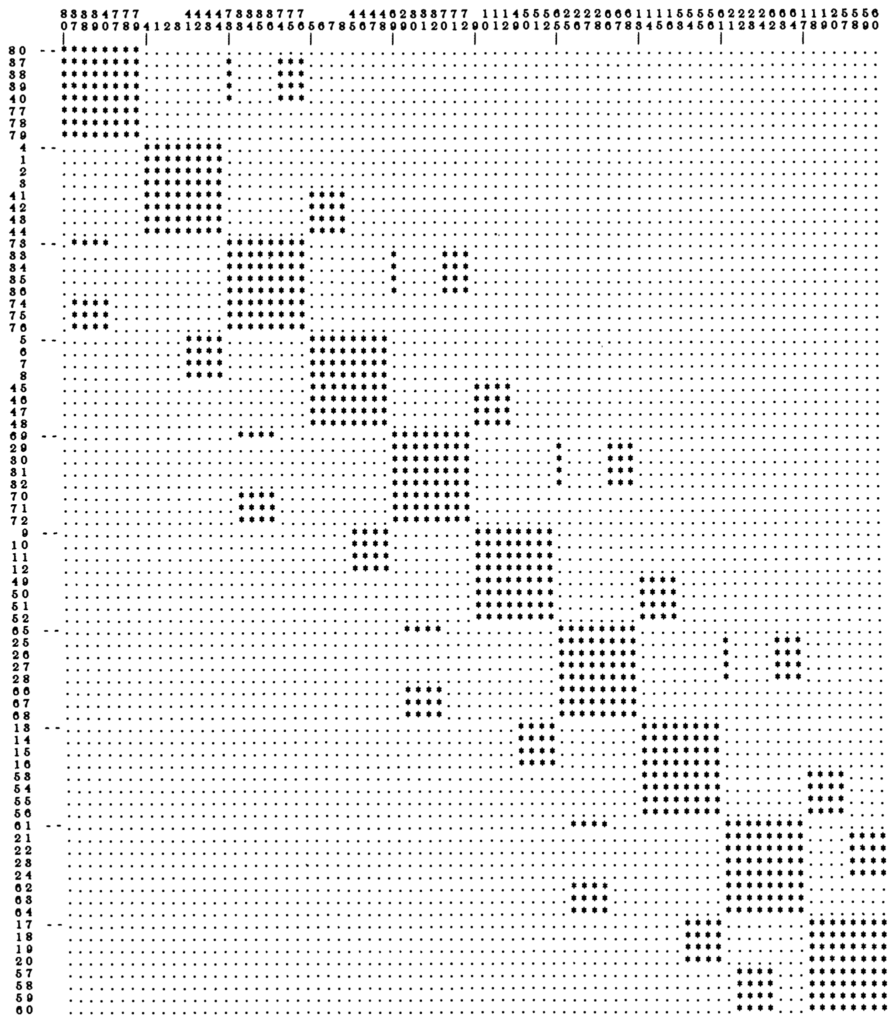


FIG. 5. *Zero-nonzero structure of "steam" permuted by* PABLO.

circuits are larger, unless the circuit is made up of small, fairly independent pieces, the human task becomes enormous and using PABLO to partition the matrix is an attractive alternative. We have observed in all these experiments and in others [12], that PABLO does effectively cluster nonzeros around the diagonal, at the expense of scattering the remaining (fewer) off-diagonal elements away from the diagonal. We include an example small enough to show the permutation produced by PABLO in detail; see also Fig. 2. Figures 4 and 5 represent the zero-nonzero structure of an 80×80 matrix called "steam" from the Harwell–Boeing sparse matrix collection [5] in its original order and permuted by PABLO, respectively. The asterisks (*) represent the nonzero entries, while the periods (.) represent the null entries. The numbers labeling the rows and columns in Fig. 5 indicate the number of the row (or column) in the original ordering.

6. Conclusion. We have presented a heuristic algorithm to permute and partition any matrix in such a way that the diagonal blocks are dense. This partition provides a convergence of classical block iterative methods that is faster than that exhibited by many other orderings. The method presented, PABLO, is based exclusively on the zero-nonzero structure of the matrix, and thus it may overlook numerical properties that could be taken into account in the partition. PABLO chooses the elements of the partition one at a time from a set of "eligible nodes." This feature makes the algorithm linear in time and space and thus of minimal cost compared to the cost of the solution of the linear system. For this reason, and based on its performance in numerous experiments, PABLO can be readily used as a general purpose preprocessor before any block iterative method is applied.

Acknowledgments. We would like to thank James T. Wilkes for his valuable help in producing the results reported in Fig. 3, Donald Erdman for his comments on the performance of PABLO on his circuit simulation codes, and Professor Stanley Eisenstat of Yale University for his pointed questions and comments, which helped improve our presentation.

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] A. BERMAN AND R. J. PLEMMONS, *Nonnegative Matrices in the Mathematical Sciences*, Academic Press, New York, 1979.

[3] I. DUFF, *MA28 – a set of Fortran subroutines for sparse unsymmetric linear equations*, Tech. Report AERE - R 8730, Computer Science and Systems Division, AERE Harwell, Oxfordshire, United Kingdom, July 1977.

[4] I. S. DUFF, A. M. ERISMAN, AND J. K. REID, *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford, 1986.

[5] I. S. DUFF, R. G. GRIMES, J. G. LEWIS, AND J. W. C. POOLE, *Sparse matrix test problems*, SIGNUM Newsletter, 17 (1982), p. 22.

[6] I. S. DUFF AND J. K. REID, *An implementation of Tarjan's algorithm for the block triangularization of a matrix*, ACM Trans. Math. Software, 4 (1978), pp. 137–147.

[7] A. GEORGE, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.

[8] A. GEORGE AND J. W. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[9] F. GUSTAVSON, *Finding the block triangular form of a sparse matrix*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, 1976, pp. 275–289.

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

[11] D. R. KINCAID AND D. M. YOUNG, *A brief review of the* ITPACK *project*, J. Comput. Appl. Math., 24 (1988), pp. 121–127.

[12] J. O'NEIL, BLITPAK: *An implementation of block-iterative methods for sparse systems of linear algebraic equations*, Tech. Report CS-1987-35, Department of Computer Science, Duke University, Durham, NC, 1987.

[13] T. C. OPPE, W. D. JOUBERT, AND D. R. KINCAID, *An overview of* NSPCG: *A nonsymmetric preconditionned conjugate gradient method*, Comput. Phys. Commun., 53 (1989), pp. 283–293.

[14] S. V. PARTER AND M. STEUERWALT, *On k-line and $k \times k$ block iterative schemes for a problem arising in three-dimensional elliptic difference equations*, SIAM J. Numer. Anal., 17 (1980), pp. 823–839.

[15] S. PISSANETZKY, *Sparse Matrix Technology*, Academic Press, London, 1984.

[16] A. POTHEN, H. D. SIMON, AND K.-P. LIOU, *Partitioning sparse matrices with eigenvectors of graphs*, Tech. Report CS-89-25, Department of Computer Science, Pennsylvania State University, University Park, PA, August 1989.

[17] D. J. ROSE, *A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations*, in Graph Theory and Computing, R. Read, ed., Academic Press, New York, 1973.

[18] R. J. TARJAN, *Depth first search and linear algebra algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.

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

[20] D. M. YOUNG, *Iterative Solution of Large Linear Systems*, Academic Press, New York, 1971.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 5, pp. 824-837, September 1990

A TIME-STEPPING ALGORITHM FOR PARALLEL COMPUTERS [*]

DAVID E. WOMBLE [†]

Abstract. Parabolic and hyperbolic differential equations are often solved numerically by time-stepping algorithms. These algorithms have been regarded as sequential in time; that is, the solution on a time level must be known before the computation of the solution at subsequent time levels can start. While this remains true in principle, it is demonstrated that it is possible for processors to perform useful work on many time levels simultaneously. Specifically, it is possible for processors assigned to "later" time levels to compute a very good initial guess for the solution based on partial solutions from previous time levels, thus reducing the time required for solution. The reduction in the solution time can be measured as parallel speedup.

This algorithm is demonstrated for both linear and nonlinear problems. In addition, the convergence properties of the method based on the convergence properties of the underlying iterative method are discussed, and an accurate performance model from which the speedup and other quantities can be estimated is presented.

Key words. time stepping, time dependent, hyperbolic, parabolic, partial differential equation (PDE), parallel computer

AMS(MOS) subject classifications. 65W05, 65M20

1. Introduction.

One route to achieving the computing power required by scientists and engineers today is through the use of parallel computers. However, to use parallel computers effectively, existing algorithms must be reexamined to take advantage of inherent parallelism, and new algorithms must be developed and analyzed.

Time-stepping methods are commonly used to numerically solve parabolic and hyperbolic partial differential equations (PDEs). In these methods, the solution to a PDE is determined at a specified set of times $t_1 < t_2 < \cdots < t_N$ in sequence beginning with t_1; the solution at one time level is completed before the solution at the next time level is started. The discretization in space can be, for example, by finite differences or by finite elements, and the discretization in time is by one-sided finite differences. The alternative approach of solving on all time levels simultaneously is not considered practical [9].

Time-stepping algorithms can be either explicit or implicit [1]. In explicit algorithms, the solution at each point in space depends only on the solutions at previous time levels, which are known. A high degree of parallelism can be achieved because the solution at each point on a time level can be calculated independently; however, explicit methods often suffer from severe restrictions on the size of the time step. In implicit algorithms, the solution at a point depends on the solutions at other points on the same time level, which are unknown. Although implicit methods do not suffer the same stepsize restriction as explicit methods, the degree of parallelism that can be achieved is reduced by the communication and synchronization requirements of solving simultaneously for each unknown at a time level [6].

Another method for the solution of time-dependent PDEs is waveform relaxation, which was originally introduced as a numerical method for circuit simulation [4]. In this method, the space variables are discretized, and the time variable remains continuous. The resulting system of initial value problems is then solved using a

[*] Received by the editors January 18, 1989; accepted for publication, September 1, 1989. This work was supported by the Applied Mathematical Sciences 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 under contract DE-AC04-76DP00789.

[†] Sandia National Laboratories, Albuquerque, New Mexico 87185.

line relaxation technique, such as Jacobi, Gauss–Seidel, or successive overrelaxation (SOR). The degree of parallelism that can be achieved depends on the relaxation technique used; however, any relaxation technique requires a substantial amount of communication and memory [5].

Finally, we mention the windowed relaxation methods described in [7] and [8]. In these methods the spatial domain is divided among the processors, and within each subdomain, each processor computes iterates on several time levels (the window) before communicating the results to other processors. In a distributed memory machine, the effect of windowing is to decrease the number of messages and increase the size of the messages, thereby increasing the processor efficiency.

In this paper, we introduce a method of parallelization for implicit time-stepping algorithms. It is applicable to a wide range of problems (linear and nonlinear) and can be coupled with a wide range of existing algorithms, including finite-element and finite-difference algorithms. Furthermore, the parallelism in our method is independent of any parallelism of the algorithm with which it is coupled. In §2, we present the parallel time-stepping method. The method is analyzed in §3, and a performance model is developed. In §4, the method is demonstrated for both linear and nonlinear problems, and the performance is compared with the predictions of the model. In §5, we summarize the paper.

2. The parallel time-stepping method.

The parallel time-stepping (PTS) method is a means by which parallelism can be introduced into a time-stepping algorithm that uses an iterative method to find the solution at each time level. Specifically, while one or more processors are computing the solution on one time level, other processors can use intermediate solutions from this time level to improve the initial guesses for the solution on later time levels. The PTS method is very general in that it can be used with either linear or nonlinear PDEs, with any discretization of a PDE, and with any iterative method for the solution at a time level. The parallelism of the PTS method is independent of any parallelism available in the iterative method with which it is coupled; however, to simplify the presentation in this section, we assume that only one processor is assigned to each time line. An example including both space and time parallelism is included in §4.

We introduce the PTS method by considering the numerical solution of a linear, parabolic PDE that has been approximated by the sequence of linear systems

$$A_n u_n = f_n + B_n u_{n-1}, \qquad n = 1, \cdots, N, \tag{1}$$

where u_n and f_n are vectors in $\mathbf{R}^m$, A_n and B_n are $m \times m$ matrices, and u_0 is given. The subscript n denotes a time level, and the vector u_n denotes the approximate solution to the PDE at a discrete set of points on that time level. We note that the vector u_{n-1} must be known before we can compute the vector u_n.

Iterative methods, such as SOR and multigrid, are often used effectively in the solution of (1). If we let $u_n^{(k)} = Q_n\left(u_n^{(k-1)}, u_{n-1}\right)$ denote the update step of such an algorithm, a serial time-stepping method for the solution of (1) can be stated as follows.

METHOD 1 (SERIAL TIME STEPPING). *Serial time stepping* (STS) *for the solution of* (1) *is given by*

```
for $n = 1, \cdots, N$ do
    set $k = 0$
```

```
        set $u_n^{(0)} = u_{n-1}$   $\left(u_1^{(0)}\text{ is the initial condition}\right)$
        until convergence do
            compute $k = k + 1$
            compute $u_n^{(k)} = Q_n\left(u_n^{(k-1)}, u_{n-1}\right)$
        end until
        set $u_n = u_n^{(k)}$
    end for
```

The number of iterations required for convergence at each time level is a function, in part, of the iteration function Q_n and the initial guess $u_n^{(0)}$. If we assume that Q_n is the best iteration function available for the solution of this problem, the only improvement that can be made is the quality of the initial guess. To this end, let us suppose that the function $R_n\left(u_n^{(0)}, u_{n-1}\right)$ can be used to refine the initial guess at time level t_n. Idle processors in a parallel computer might then use R_n to improve the initial guesses on time levels $t_{n+1}, \cdots, t_N$ while the solution is being computed on time level t_n. For example, suppose we use three processors to solve on three time levels. Processor 1 solves the problem at the first time level by repeated evaluations of the iteration function Q_1, and after each evaluation of Q_1, sends the approximate solution to processor 2. Processor 2 uses this approximate solution at time level 1 to generate a new initial guess using the function R_2. This initial guess is then sent to processor 3, which treats it as an approximate solution at time level 2 and generates a new initial guess for time level 3 using R_3. After processor 1 has solved the problem on time level 1, processor 2 begins the solution process at time level 2 using its (improved) initial guess and the iteration function Q_2 while processor 3 continues to improve its initial guess using R_3. Finally, after processor 2 has solved the problem at time level 2, processor 3 solves the problem at time level 3 using Q_3.

Pseudocode for the PTS method is given below. The first loop in the pseudocode contains the evaluation of the function R_n and is executed until the solution on the previous time level has converged. The second loop corresponds to the loop in the serial time-stepping method and is executed until the solution on the current time level has converged.

METHOD 2 (PARALLEL TIME STEPPING). *If N processors are available for the solution of* (1), *then the parallel time-stepping* (PTS) *method for processor n is given by*

```
set $k = 0$
set $u_n^{(0)}$ to the initial condition
if $n \neq 1$ then
    until convergence on processor $n - 1$ do
        compute $k = k + 1$
        receive $u_{n-1}^{(k-1)}$ from processor $n - 1$
        compute $u_n^{(k)} = R_n\left(u_n^{(k-1)}, u_{n-1}^{(k-1)}\right)$
        send $u_n^{(k)}$ to processor $n + 1$
    end until
end if
set $u_{n-1} = u_{n-1}^{(k)}$
set $l = 0$
set $u_n^{(0)} = u_n^{(k)}$
until convergence do
    compute $l = l + 1$
    compute $u_n^{(l)} = Q_n\left(u_n^{(l-1)}, u_{n-1}\right)$
    send $u_n^{(l)}$ to processor $n+1$
end until
set $u_n = u_n^{(l)}$
```

There are many possible choices for the function R_n. For example, R_n might correspond to a multigrid algorithm in which a different combination of grids is used than for Q_n. (This would be useful because iterative methods eliminate different frequencies at different rates and different frequencies can propagate forward in time at different speeds.) One practical choice for the iteration function R_n is to set it equal to Q_n. There are several reasons for this. First, we assume that Q_n was chosen because of desirable convergence properties for the problem to be solved. Second, because the $R_{n+1}, \cdots, R_N$ are used concurrently with Q_n, setting $R_n = Q_n$ results in a load balanced algorithm. (Note that in many cases, the amount of work required for one application of the functions Q_n and R_n does not depend on n.) Third, the task of implementing the algorithm is simplified. Throughout the remainder of the paper, we will take $R_n = Q_n$.

In practice, we have P ($< N$) processors available for the solution of (1). In this case, processor p begins by computing the solution at time level t_p. When work has been completed on this level, it begins refining the initial guess on time level t_{P+p} and eventually computes the solution there. This process continues until the solution has been computed on each of the N time levels.

We note from the pseudocode above that processor n cannot begin work until a message has been received from processor $n-1$, which occurs after processor $n-1$ has completed one iteration. To formalize this, we introduce the concept of the delay at time level n, d_n, which we define to be the number of iterations that processor $n-1$ completes before processor n starts. For the pseudocode listed above, $d_1 = 0$ and $d_n = 1$, $n = 2, \cdots, N$. In the case of P ($< N$) processors, the delays d_n are unknown a priori for $n = P+1, \cdots, N$.

Even though we have developed the PTS method for a linear, parabolic PDE with a finite-difference discretization, it is clear that the method can be used to parallelize any time-stepping algorithm, including those for nonlinear problems, those for hyperbolic PDEs, and those that use finite-element discretizations. The PTS method is demonstrated for a variety of problems in §4.

3. Analysis.

The parallel time-stepping method is very general. We have not specified the type of equation, the method of discretization, or the iterative solution algorithm. Thus, no one proof of convergence of the iteration $u_n^{(k)} = Q_n\left(u_n^{(k-1)}, u_{n-1}\right)$, $k = 1, 2, \cdots$, can be constructed. However, for a linear PDE, the convergence of the serial time-stepping method (for a choice of discretization and iterative solution algorithm) implies the convergence of the parallel time-stepping method because the iteration defined by Q_n will converge for any initial guess.

We can develop a model for the behavior of the parallel time-stepping method for linear PDEs. (Models can also be developed for nonlinear problems, but they are highly problem dependent.) We assume that a linear PDE has been reduced to the form

$$A_n u_n = f_n + B_n u_{n-1}, \qquad n = 1, \cdots, N, \tag{2}$$

where u_n and f_n are vectors in $\mathbf{R}^m$, A_n and B_n are $m \times m$ matrices, u_0 is known, and A_n is nonsingular. We let

$$u_n^* = A_n^{-1}(f_n + B_n u_{n-1})$$

be the solution to (2) at time level n and define an iterative method for obtaining u_n^* by

$$u_n^{(k)} = Q_n\left(u_n^{(k-1)}, u_{n-1}\right). \tag{3}$$

To guarantee the convergence of $u_n^{(k)}$ to u_n^*, we require that Q_n satisfy the Lipschitz condition

$$\|Q_n(u,v) - Q_n(u_n^*, v)\| \le \beta_n \|u - u_n^*\|, \qquad \beta_n < 1,$$

for all u and v in $\mathbf{R}^m$ and some norm $\|\cdot\|$ on $\mathbf{R}^m$. These definitions form the traditional framework for the study of the STS method. To study the PTS method, we adopt the convention that $u_0^{(k)} = u_0$, $k = 1, \cdots, \infty$, and define

$$u_n^{(k),*} = \begin{cases} A_n^{-1}\left(f_n + B_n u_{n-1}^{(k+d_n)}\right), & k = 1, \cdots, I_{n-1} - d_n, \\ A_n^{-1}\left(f_n + B_n u_{n-1}^{(I_{n-1})}\right), & k > I_{n-1} - d_n, \end{cases}$$

where d_n is the delay defined in the previous section, and I_{n-1} is the number of iterations required for convergence of the iteration (3) to the solution of (2) at time level $n-1$. In the notation of Method 2, I_n is the number of times R_n is evaluated plus the number of times Q_n is evaluated. The vector $u_n^{(k),*}$ is thus the solution to (2) with the true solution at time level $n-1$ replaced by the most recent iterate, and we note that $u_n^{(k),*} = Q_n(u_n^{(k),*}, u_{n-1}^{(k+d_n)})$. Finally, we define $e_n^{(k)}$ by

$$e_n^{(k)} = \begin{cases} \|u_n^{(k)} - u_n^{(k),*}\|, & k = 1, \cdots, I_n, \\ 0, & k > I_n. \end{cases}.$$

For k between 1 and I_n, $e_n^{(k)}$ is the norm of the difference between the iterate $u_n^{(k)}$ and the "apparent" true solution $u_n^{(k),*}$, which we refer to as the apparent error. For $k > I_n$, we set $e_n^{(k)} = 0$ to simplify error bound on later time levels.

We now derive a bound for $e_n^{(k)}$. Applying the Lipschitz condition on Q_n for $n = 1$ yields

$$e_1^{(k)} \le \beta_1^k e_1^{(0)},$$

and for $n > 1$,

$$\begin{aligned} e_n^{(k)} &\le \beta_n \|u_n^{(k-1)} - u_n^{(k),*}\| \\ &\le \beta_n \left(\|u_n^{(k-1)} - u_n^{(k-1),*}\| + \|u_n^{(k-1),*} - u_n^{(k),*}\|\right) \\ &\le \beta_n \left(e_n^{(k-1)} + \|A_n^{-1}B_n\| \, \|u_{n-1}^{(k-1+d_n)} - u_{n-1}^{(k+d_n)}\|\right) \\ &\le \beta_n \left(e_n^{(k-1)} + (1+\beta_{n-1})\|A_n^{-1}B_n\| \, \|u_{n-1}^{(k-1+d_n)} - u_{n-1}^{(k+d_n),*}\|\right). \end{aligned}$$

Applying the above calculations recursively yields

$$(4)\quad e_n^{(k)} \leq \beta_n \left(e_n^{(k-1)} + \sum_{i=1}^{n-1} \left(e_i^{\left(k-1+\sum_{j=i}^{n-1} d_{j+1}\right)} \prod_{j=i}^{n-1} (1+\beta_j) \|A_{j+1}^{-1} B_{j+1}\| \right) \right).$$

We note from (4) and the definition of $e_n^{(k)}$ that if $d_n = I_{n-1}$, then we recover the traditional error bounds for the STS method. Using induction, we can also conclude from (4) that the PTS method converges (for linear problems).

We also note from (4) that the effect of the error on one time level can be magnified at all later time levels on which the computation is proceeding simultaneously. For many practical problems, this magnification factor is greater than one. Hence, our upper bound on the error allows the possibility that the PTS method on N processors requires more time to solve a problem than does the STS method on one processor.

The upper bound on the error (4) is not tight; however, the error can be approximated. Because, the iterates, $u_{n-1}^{(k)}$, asymptotically approach u_{n-1}^* along a vector lying in the subspace spanned by the eigenvectors corresponding to the largest eigenvalue, the distance between consecutive iterates asymptotically approaches $1-\beta_{n-1}$. Replacing the term $1+\beta_{n-1}$ in the upper bound with $1-\beta_{n-1}$ yields

$$\begin{aligned} e_n^{(k)} &\leq \beta_n \left(e_n^{(k-1)} + \|A_n^{-1} B_n\| \, \|u_{n-1}^{(k-1+d_n)} - u_{n-1}^{(k+d_n)}\| \right) \\ &\approx \beta_n \left(e_n^{(k-1)} + (1-\beta_{n-1}) \|A_n^{-1} B_n\| e_{n-1}^{(k-1+d_n)} \right). \end{aligned}$$

Thus, $e_n^{(k)}$ is approximated by $\sigma_n^{(k)}$, the solution to the recursion

$$(5)\qquad \sigma_n^{(k)} = \begin{cases} \beta_n \sigma_n^{(k-1)} + \alpha_n \sigma_{n-1}^{(k-1+d_n)}, & \sigma_{n-1}^{(k-1+d_n)} \geq \epsilon, \\ 0 & \text{otherwise} \end{cases}$$

for $k = 1, 2, \cdots$ and $n = 1, 2, \cdots$ with the initial conditions

$$\sigma_0^{(k)} = 0, \qquad k = 0, 1, \cdots,$$

$$\sigma_n^{(0)} = \|u_n^* - u_{n-1}^*\| + \gamma_n \sigma_{n-1}^{(d_n - 1)}, \qquad n = 1, 2, \cdots,$$

where

$$\alpha_n = \beta_n (1-\beta_{n-1}) \|A_n^{-1} B_n\|,$$

$$\gamma_n = 1 + \|A_n^{-1} B_n\|,$$

and ϵ corresponds to the convergence criterion. This recursion can be solved in closed form; however, this form does not yield additional information. Instead, we will evaluate the recursion relation numerically for specific cases. The approximations for the error will be used to predict speedups, which will be compared with experimental results in the next section.

The delays can be used to generalize the model to the case of P ($< N$) processors. For example, to model the one processor case (STS method), we can set d_n equal to I_n, the minimum k such that $\sigma_n^{(k)} = 0$. To model the P processor case, the delays on the first P time levels are arbitrary, and the delay at time n ($> P$) is chosen so that computation does not start until the solution on line $n - P$ has been completed. We note that the maximum number of processors that can be used without at least one processor being idle at all times must satisfy

$$P \leq I_n \bigg/ \sum_{j=n+1}^{n+P} d_j \tag{6}$$

for all n between 1 and $N - P$. This relation states that the processor assigned to time level n cannot complete its calculations before each of the remaining processors is assigned a time level and begins iterating.

The most common performance measure is speedup. It is normally defined as the time required for one processor to solve a problem divided by the time required for P processors to solve the same problem. The time to solve a problem using either Method 1 or Method 2 is proportional to the "effective" number of evaluations of Q_n, that is, the number of evaluations of Q_n that are not overlapped with computations on previous time levels. The effective number of evaluations of Q_n is given by $\sum_{n=1}^{N} E_n$, where $E_n = I_n - I_{n-1} + d_n$. If we denote by $I_n(P)$ the number of iterations and by $E_n(P)$ the effective number of iterations required for convergence at time level n and by $d_n(P)$ the delay at time level n in the P processor case, $S(P)$, the speedup on P processors, is given by

$$\begin{aligned} S(P) &= \left(\sum_{n=1}^{N} I_n(1)\right) \bigg/ \left(\sum_{n=1}^{N} E_n(P)\right) \\ &= \left(\sum_{n=1}^{N} I_n(1)\right) \bigg/ \left(I_N(P) + \sum_{n=1}^{N} d_n(P)\right). \end{aligned} \tag{7}$$

For some problems, $I(1) = I_1(1) = \cdots = I_N(1)$ are constant. In this case, there is a "steady-state" solution of (5) in which $I(P) = I_1(P) = \cdots = I_N(P)$ are constant, and $d(P) = I(P)/P$, and the "steady-state" speedup is given by

$$SS(P) = \lim_{N \to \infty} S(P) = P \times I(1) \Big/ I(P). \tag{8}$$

The steady-state speedup is an asymptotic value for the speedup (as the number of time levels at which the solution is desired increases). The "transient" nature of the speedup for a small number of time levels is due to the fact that none of the iterations at the first time level can be overlapped with iterations at previous time levels.

We now look at the effect of the parameters in the model on the performance of the PTS method. The delays, the terms $\|A_n^{-1}B_n\|$, and the convergence factors β_n have the largest effect, while the effect of the other parameters is minimal. The results presented in the remainder of this section are obtained by numerically evaluating the recursion relation (5). The values of the parameters used in the evaluation of (5) are close to values seen in many applications, and the effects shown in the remainder of this section are observed over a range of values for the parameters.

The delay is a function of both hardware and software. It depends on the number of processors, the number of iterations that each processor requires for convergence, and the time to "start up" a processor on a new time level. In almost all cases, we want the minimum delay possible; nevertheless, it is instructive to consider the effect of the delay on the number of iterations. As the delay d_n is decreased, the error introduced as the result of error at the previous time level is increased, and we expect that the number of iterations required for convergence I_n will increase. However, the numerical evaluation of (5) indicates that the "effective number of iterations," $E_n = I_n - I_{n-1} + d_n$, will be reduced. This is shown in Fig. 1.

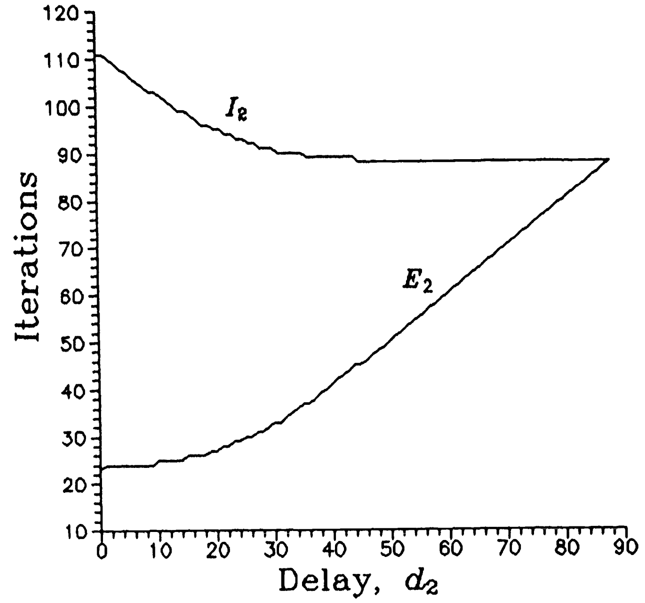


FIG. 1. *The effect of the delay d_2 on the number of iterations I_2 and the effective number of iterations E_2. ($\beta_2 = .9$, $\alpha_2 = .09$, $\|u_2^* - u_1^*\| = .1$, $\epsilon = .00001$.) Note that $I_1 = 88$.*

The terms $\|A_n^{-1}B_n\|$ are determined by the PDE and the method of discretization. We see from (5) that increasing the value of $\|A_n^{-1}B_n\|$ magnifies the effect of the error at the previous time level. The result is that the number of iterations required at each time level increases, the delays d_n ($n > P$) increase, and the speedup decreases. Noting equation (8) and the effect of the delays on the number of iterations shown in Fig. 1, we expect the decrease in speedup to be much more severe for a large number of processors. This effect is shown in Fig. 2. There is a slight "staircase" nature to the curves, which is due to the fact that d_n and I_n are integers.

The convergence factors β_n are determined by the iterative method chosen and affect both the number of iterations required for convergence and the magnification of the error on the previous time level. As β_n approaches one, errors from previous time levels are introduced faster than they can be eliminated. Thus, the effective number of iterations increases, and we expect the speedup to decrease. On the other hand, as β_n approaches zero, the number of iterations required for convergence decreases, and we see from equation (6) that the number of processors that can be used effectively decreases. Thus, we expect a decrease in the speedup. The overall effect of changing β_n on the steady-state speedup is shown in Fig. 3. We note that the speedup achieves its maximum on the interior of the interval $(0, 1)$, and that the PTS method performs best with relatively good iterative algorithms. As before, the staircase nature of the curves is due to the fact that d_n and I_n are integers.

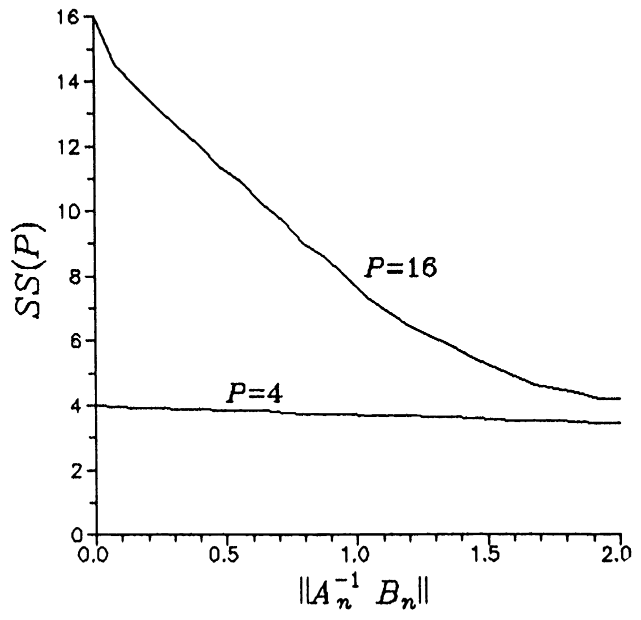


FIG. 2. *The effect of* $\|A_n^{-1}B_n\|$ *on the steady-state speedup.* ($\beta_n = .9$, $\|u_n^* - u_{n-1}^*\| = .1$, $\epsilon = .00001$, $P = 16$.)

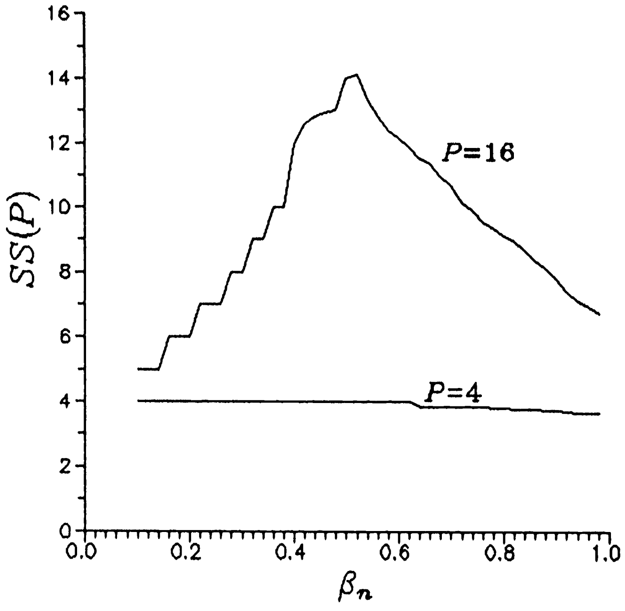


FIG. 3. *The effect of* β_n *on the steady-state speedup.* ($\|A_n^{-1}B_n\| = 1$, $\|u_n^* - u_{n-1}^*\| = .1$, $\epsilon = .00001$, $P = 16$.)

4. Experimental results. In this section, we present three examples of the PTS method. The first example is a parabolic PDE and an iterative method for which the parameters in the model, equation (5), can be calculated analytically. This allows direct comparison of the performance of the PTS method with the predictions of the model. The second example is a nonlinear, parabolic PDE with multigrid as the underlying iterative method. This example demonstrates that the PTS method is applicable to a wide variety of problems. In the third example, we demonstrate that the PTS method can be effectively coupled with a parallel implementation of an iterative algorithm at each time level. All numerical results were obtained on the NCUBE/ten hypercube. For ease of programming, we assume that the number of processors to be used is a power of two, although this is not a requirement for the PTS method.

EXAMPLE 1. The first example is the PDE

$$\begin{gathered} -\frac{\partial^2 u}{\partial x^2} + \frac{\partial u}{\partial t} = 3, \qquad (x,t) \in (0,1)\times(0,5), \\ u(x,0) = 0, \qquad x \in (0,1), \\ u(0,t) = 3t = u(1,t), \qquad t \in (0,5). \end{gathered} \tag{9}$$

We let $\Delta x = 1/64$ and $\Delta t = 5/200$ and replace (9) with the finite-difference approximation

$$\begin{aligned} \frac{-u_{i+1,n} + 2u_{i,n} - u_{i-1,n}}{\Delta x^2} + \frac{u_{i,n} - u_{i,n-1}}{\Delta t} &= 3, \\ i &= 1,\cdots,63, \quad n = 1,\cdots,200, \end{aligned}$$

$$u_{i,0} = 0, \qquad i = 1,\cdots,63,$$

$$u_{0,n} = 3n\Delta t = u_{M,n}, \qquad n = 1,\cdots,200.$$

This yields a linear system of the form (1), which we solve with SSOR iteration with the near optimal relaxation parameter $\omega = 1.8$. The parameters needed to evaluate equation (5) can be calculated analytically. They are

$$\beta_n = .83, \quad \|A_n^{-1}B_n\| = .80, \quad \|u_n^* - u_{n-1}^*\| = .075, \qquad n = 1,\cdots,200.$$

For the convergence criterion, we let $\epsilon = 1 \times 10^{-6}$. Convergence can be checked explicitly because the true solution to (9) is known.

The problem was solved numerically using different numbers of processors. The results and the predictions of the model are shown in Table 1. Note that problem (9) satisfies the requirements necessary to compute the steady-state speedup. As was stated in the previous section, the steady-state speedup is an asymptotic value for the speedup as the number of time levels on which the solution is desired increases.

We make two observations based on Table 1. The first is that with only 200 time steps, we cannot use more than 200 processors. The second observation is that the model is most accurate for $P \ll N = 200$. One reason is that the delays are larger, and small errors in modeling the delay have a smaller effect. (We recall that the delays

TABLE 1
Comparison of the performance of the PTS *method for Example* 1 *with the predictions of the model. The predictions are in parentheses.*

P	$\sum_{n=1}^{200} E_n$		$S(P)$		$SS(P)$	
1	12,000	(12,200)	1.00	(1.00)	1.00	(1.00)
2	6,118	(6,121)	1.96	(1.99)	2.00	(2.00)
4	3,168	(3,201)	3.79	(3.81)	3.90	(3.87)
8	1,847	(1,915)	6.49	(6.37)	6.53	(6.78)
16	1,234	(1,319)	9.72	(9.25)	9.50	(10.8)
32	872	(1,137)	13.7	(10.7)	13.9	(15.2)
64	634	(1,137)	18.7	(10.7)	19.1	(20.0)
128	489	(1,137)	24.5	(10.7)	27.6	(23.6)

are dependent, in part, on the hardware and must be modeled.) Another reason is that we used the approximation

$$\|Q_n(u,v) - Q_n(u_n^*,v)\| \approx \beta_n\|u - u_n^*\|, \qquad \beta_n < 1,$$

which is most accurate for a large number of iterations (with β_n equal to the spectral radius of Q_n). For $P = 128$, there is an average of less than three effective iterations per time step.

EXAMPLE 2. As a second example, we choose Burgers' equation:

$$\nu\frac{\partial^2 u}{\partial x^2} - u\frac{\partial u}{\partial x} - \frac{\partial u}{\partial t} = 0, \qquad (x,t) \in (0,1)\times(0,5),$$

$$u(x,0) = \sin(\pi x), \qquad x \in (0,1), \tag{10}$$

$$u(0,t) = 0 = u(1,t), \qquad t \in (0,5).$$

We let $\Delta x = 1/128$ and $\Delta t = 5/200$ and replace (10) with the finite-difference approximation to get

$$\nu\frac{u_{i+1,n} - 2u_{i,n} + u_{i-1,n}}{\Delta x^2} - u_{i,n}\frac{u_{i,n} - u_{i-1,n}}{\Delta x} - \frac{u_{i,n} - u_{i,n-1}}{\Delta t} = f_{i,n},$$

$$i = 1,\cdots,127, \quad n = 1,\cdots,200, \tag{11}$$

$$u_{i,0} = \sin(i\pi\Delta x), \qquad i = 0,\cdots,128,$$

$$u_{0,n} = 0 = u_{128,n}, \qquad n = 1,\cdots,200.$$

We note that upwind differencing has been used for the term $\partial u/\partial x$.

As an iterative method for the solution of (11), we choose multigrid iteration with a weighted Jacobi smoothing step and a weighting factor of $\omega = .95$. Because multigrid iteration requires a linear equation, we delay $u_{i,n}$ in the nonlinear term by one cycle. We consider the iterations to have converged if the residual is less than 1.0×10^{-5}. Table 2 shows the results obtained by running the PTS algorithm for

TABLE 2
The performance of the PTS *algorithm for Example* 2 *with* $\nu = .01$.

P	$\sum_{n=1}^{200} E_n$	$S(P)$
1	5,346	1.00
2	2,732	1.96
4	1,341	3.99
8	765	7.00
16	570	9.38
32	505	10.6
64	487	11.0
128	487	11.0

this problem with $\nu = .01$ on different numbers of processors. There are no predicted results because the problem is nonlinear.

We note that the performance of the PTS algorithm for Example 2 is somewhat worse than that for Example 1. The reason for this is that fewer iterations are required at each time step in Example 2. Thus, fewer processors can be used effectively. In general, for problems that have steady-state solutions, the number of iterations decreases as n increases (as in Example 2), and as a result, fewer processors can be used effectively at later time levels.

EXAMPLE 3. The third example is a linear, parabolic PDE that arises in the study of grain-boundary diffusion [3]. The dimensionless equation with parameters approximating the diffusion of chromium in gold is

$$0.01\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)=\frac{\partial u}{\partial t}, \qquad (x,y,t)\in(0,1)\times\left(0,\frac{1}{2}\right)\times(0,1), \tag{12}$$

with boundary conditions

$$\frac{\partial u}{\partial t}(x,0,t)=\frac{\partial^2 u}{\partial x^2}(x,0,t)+0.1\frac{\partial u}{\partial y}(x,0,t), \qquad x\in(0,1), \quad t\in(0,1),$$

$$\frac{\partial u}{\partial x}\left(x,\frac{1}{2},t\right)=0, \qquad x\in(0,1), \quad t\in(0,1),$$

$$u(0,y,t)=1, \qquad y\in\left(0,\frac{1}{2}\right), \quad t\in(0,1),$$

$$u(1,y,t)=0, \qquad y\in\left(0,\frac{1}{2}\right), \quad t\in(0,1),$$

and initial conditions

$$u(x,y,0)=0, \qquad x\in(0,1], \quad y\in\left(0,\frac{1}{2}\right),$$

$$u(0,y,0)=1, \qquad y\in\left(0,\frac{1}{2}\right),$$

TABLE 3
The performance of the PTS *algorithm for Example* 3. P_x *and* P_y *are the numbers of processors in the* x *and* y *direction, respectively, and* P_t *is the number of time levels on which iterations are carried out simultaneously. The total number of processors used is* $P_x \times P_y \times P_t$. *Run times are in seconds, and speedups are given in parentheses.*

	P_t				
$P_x \times P_y$	1	2	4	8	16
1×1	467. (1.00)	264. (1.77)	178. (2.62)	145. (3.22)	124. (3.77)
2×2	135. (3.46)	76.1 (6.13)	52.0 (8.98)	42.6 (11.0)	35.9 (13.0)
4×4	47.3 (9.87)	26.6 (17.6)	18.4 (25.4)	15.0 (31.1)	12.6 (37.1)
8×8	24.1 (19.4)	13.8 (33.8)	9.44 (49.5)	7.73 (60.4)	6.64 (70.3)

$$u(x,0,0) = 1 - x, \qquad x \in (0,1).$$

We let $\Delta x = 1/32$, $\Delta y = 1/64$, and $\Delta t = 1/32$, and replace (12) with an implicit finite-difference approximation (implicit Euler difference in time, central differences in space where possible, one-sided differences otherwise) to get a system of linear equations of the form (1). As a parallel iterative algorithm for the solution at each time level, we choose Jacobi iteration because of its inherent parallel nature and consider the iterations to have converged when the residual is less than 1.0×10^{-3}. Table 3 shows the run times and speedups for runs with various combinations of processors in the space and time directions.

Even though Jacobi iteration is considered a highly parallel algorithm, a problem size of 32×32 is small and communication overhead is significant. We see from Table 3 that if only a small number of processors are available for the solution of (12), then they are most effectively used by the parallel Jacobi algorithm to solve at one time level. However, if a large number of processors are available, then they are most effectively used when the Jacobi iteration is coupled with the PTS method. This effect would be more pronounced if we had chosen an iterative method with a less efficient parallel implementation, such as multigrid or SOR.

We can also see from Table 3 that the total speedup is approximately equal to the speedup obtained in the space variables times the speedup obtained in the time variable.

5. Summary. In this paper, we have presented a technique by which parallelism in the time direction can be introduced into implicit time-stepping algorithms. The attraction of the technique is that it can be coupled with a wide variety of algorithms and that the parallelism introduced in the time direction is independent of any parallelism in space. Thus, the number of processors that can be efficiently applied to the solution of a time-dependent PDE is increased by at least an order of magnitude.

We also presented a performance model of the method for linear problems. Based on this model, we were able to predict the effect of algorithm parameters, such as the convergence rate and the number of processors used, on the speedup. In §4, this model was found to be in good agreement with the actual performance of the method.

Finally, we demonstrated the PTS method for linear and nonlinear problems and for three common iterative methods. We found that the method was very effective for a small number of processors and remained effective while the number of processors was less than the number of time levels on which the solution was desired and less than the number of iterations required for convergence on a time level. We also demonstrated that the PTS method can be effectively coupled with parallel iterative algorithms for the solution at each time level.

Acknowledgments. The author thanks E. F. Brickell and D. E. Amos of Sandia National Laboratories for their help in the analysis of equation (5).

REFERENCES

[1] W. F. AMES, *Numerical Methods for Partial Differential Equations*, Academic Press, New York, 1977.

[2] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.

[3] P. H. HOLLOWAY, D. E. AMOS, AND G. C. NELSON, *Analysis of grain-boundary diffusion in thin films: Chromium in gold*, J. Appl. Phys., 47(1976), pp. 3769–3775.

[4] E. LELARASMEE, A. RUHELI, AND A. L. SANGIOVANNI-VINCENTELLI, *The waveform relaxation method for the time domain analysis of large scale integrated circuits*, IEEE Trans. Computer-Aided Design, 1(1982), pp. 131–145.

[5] A. R. NEWTON AND A. L. SANGIOVANNI-VINCENTELLI, *Relaxation-based electrical simulation*, SIAM J. Sci. Statist. Comput., 4(1983), pp. 485–524.

[6] J. M. ORTEGA AND R. G. VOIGT, *Solution of Partial Differential Equations on Vector and Parallel Computers*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1985.

[7] J. H. SALTZ AND V. K. NAIK, *Towards developing robust algorithms for solving partial differential equations on* MIMD *machines*, Parallel Comput., 6(1988), pp. 19–44.

[8] J. H. SALTZ, V. K. NAIK, AND D. M. NICOL, *Reduction of the effects of the communication delays in scientific algorithms on message passing* MIMD *architectures*, SIAM J. Sci. Statist. Comput., 8(1987), pp. s118–s134.

[9] G. STRANG AND G. J. FIX, *An Analysis of the Finite Element Method*, Prentice-Hall, Englewood Cliffs, NJ, 1973.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 5, pp. 838–858, September 1990

THE EFFECT OF TIME CONSTRAINTS ON SCALED SPEEDUP*

PATRICK H. WORLEY†

Abstract. Gustafson, Montry, and Benner introduced the concept of scaled speedup to characterize the capabilities of distributed-memory multiprocessors. They argued that, for a fixed-size problem, the behavior of the speedup of an algorithm as a function of the number of processors, the *speedup curve*, can be too pessimistic a measure of a multiprocessor architecture. Instead, they measured the speedup of algorithms when the size of the corresponding problem grew with the number of processors. They referred to the resulting function as the *scaled speedup curve.*

The scaled speedup curve is a function of how the size of the problem is allowed to grow. In this paper, it is demonstrated that allowing the size of a problem to grow to fill the available memory can produce dramatically different results from allowing the size of a problem to grow subject to satisfying an upper bound on the execution time. In particular, if a constraint on the execution time is enforced, then the scaled speedup curve is often very similar to the speedup curve for a fixed-size problem. It is shown that no more than 50 processors can be used efficiently for some common problems in scientific computation when using the current generation of distributed-memory multiprocessors. For other problems, it is shown that the scaled speedup curve indicates that massively parallel computers will be useful even if the execution time is constrained. In all of the cases examined, a meaningful interpretation of the scaled speedup curve depends on a constraint on the execution time.

Key words. limits on parallelism, linear partial differential equations, massively parallel computation, problem scaling, scaled speedup

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

1. Introduction.

Distributed-memory MIMD multiprocessors[1] with a moderate number of processors (< 100) have proven to be cost-effective computer architectures for solving many of the compute-intensive problems in scientific computing [6], [11], [12]. Multiprocessors with orders-of-magnitude larger numbers of processors can also be cost-effective [9], [16], but it is unclear whether this sort of massive parallelism is as useful as the more moderate amount of parallelism currently being exploited.

The tool most commonly used to argue for or against multiprocessors with very large numbers of processors is the *speedup curve* [2], [17]. The speedup of an algorithm on a multiprocessor with P identical processors is the ratio

$$S \equiv \frac{T_1}{T_P}, \tag{1}$$

where T_1 is the execution time of a serial implementation of the algorithm on one of the processors and T_P is the execution time of a parallel implementation of the algorithm that uses all P of the processors. The speedup curve is the graph of the speedup as a function of the number of processors. This curve is a measure of the performance of a family of multiprocessors, one for each number of processors. When the processors are identical across the family and the interconnection topology varies in a natural way as a function of the number of processors, then this family represents

* Received by the editors December 27, 1988; accepted for publication (in revised form) November 9, 1989. This research was supported by the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy under contract DE-AC05-84OR21400 with Martin Marietta Energy Systems Inc.

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

[1] *M*ultiple *I*nstruction *M*ultiple *D*ata is one category of Flynn's multiprocessor taxonomy [5]. If a computer is an MIMD multiprocessor, then both the instruction a processor is executing and the data it is using can be different from those of other processors at any given moment.

the scaling of a multiprocessor architecture. The maximum value reached by the curve indicates the maximum speedup achievable by the associated architecture for the given algorithm.

For a fixed-size problem, there is a law of diminishing returns when using an increasing number of processors to calculate the solution. There is only so much work to be done, and rarely can all of it be done in parallel. Details of this type of behavior can be derived from the upper bound on the speedup described by Amdahl in [1]:

$$S \leq \frac{1}{s + (1-s)/P}. \tag{2}$$

P is the number of processors and s is the fraction of the work that cannot be parallelized. Inequality (2) is referred to as Amdahl's law.

Two implications can be drawn from Amdahl's law. First, if s is nonzero, then the speedup is bounded from above by $1/s$, independent of the number of processors in the multiprocessor. Second, if the right-hand side of Amdahl's law is a good model of the speedup curve, then only a relatively few processors are needed to achieve most of the maximum speedup. For example, if $s = .1$, then the speedup cannot exceed 10, and 36 processors achieve 80 percent of this value.

The right-hand side of Amdahl's law is not a good model of the speedup curve for MIMD multiprocessors. The performance of a parallel algorithm is degraded whenever fewer than P processors are busy during the execution of the algorithm, not just when only one processor is busy. Moreover, there is often an overhead involved when exploiting parallelism, and processors must execute instructions that do not exist in the serial implementation. But generalizations of Amdahl's law have been developed by numerous researchers that lead to the same type of conclusions implied by Amdahl's law. For examples, see [17, pp. B29-B34].

Recently, Gustafson, Montry, and Benner argued that the analysis of fixed-size problems can be misleading when evaluating a multiprocessor architecture [9]. Instead of the fixed-size analysis described above, they proposed examining the speedup curve when the size of the problem increases with the number of processors.[2] They referred to this as the *scaled speedup curve.* The implications of the fixed-size analysis do not change unless s decreases as a function of P, but this is exactly what is observed in practice for many algorithms [9]. The theoretical analyses of both Amdahl and Fox, described by Messina in [17], support these results for many algorithms in scientific computation *if the problem size increases sufficiently fast as a function P.*

In this paper we examine the effect of two different assumptions on how the problem size grows. The first assumption, the one used in [9], allows the size of the problem to grow to fill the available memory. We will refer to this as the *memory-constrained* case. The second assumption allows the size of the problem to grow subject to an upper bound on the execution time. We will refer to this as the *time-constrained* case. In previous work [23], [24] we proved that the execution time will grow without bound as a function of the problem size, independent of the number of processors and of the algorithm used, for certain common problems in scientific computation. We also argued that similar conclusions apply to most problems in scientific computation. For example, the result holds whenever (a) the solution to a problem includes some scalar quantity that cannot be calculated exactly (using available information) without using an infinite amount of data, (b) the size of the

[2] A similar approach was used by Moler in [18].

problem is allowed to increase only if the error in the calculated solution decreases, and (c) if the size of the problem grows unboundedly, then the error goes to zero. When these assumptions are satisfied, there will be a largest problem that can be solved and still satisfy a given time constraint, and the scaled speedup for the time-constrained case will not exceed some constant determined by this largest problem. The memory-constrained case is not limited in this fashion.

The theoretical analysis in [23] is relatively tight, but the lower bounds on the execution time can be quite small because they hold for an optimal parallel algorithm and an unlimited number of processors. To determine whether this distinction between the memory-constrained case and the time-constrained case has practical import, we examine scaled speedup curves for some simple algorithms used to approximate the solution of model linear partial differential equations (PDE). In §2 we introduce the multiprocessor model used to describe and analyze the parallel algorithms. In §3 we analyze an explicit finite-difference algorithm for a scalar hyperbolic PDE in two space dimensions. We then generalize the results to the corresponding problems in other space dimensions. In §4 we analyze an implicit finite-difference algorithm for a scalar elliptic PDE in one space dimension. We also discuss how to generalize these results to the corresponding problems in higher dimensions.

Another tool used to evaluate the performance of an algorithm on a multiprocessor is the *efficiency*,

$$E \equiv \frac{S}{P}. \tag{3}$$

It measures the fraction of the maximum possible speedup that is achieved. The efficiency is useful when deciding how best to allocate processors among independent programs in order to maximize throughput. The efficiency and scaled efficiency curves are defined in an analogous manner to the speedup and scaled speedup curves. We will also mention the effect of time and memory constraints on the scaled efficiency.

2. Multiprocessor model.

To facilitate comparison with the empirical results of Gustafson, Montry, and Benner [9], the following multiprocessor model is based on the NCUBE family of hypercube multiprocessors [4]. The model is not too different from distributed-memory MIMD multiprocessors available from other manufacturers [4], and similar conclusions can be drawn for these architectures.

The following general assumptions are basic to the analysis in this paper.

1) We assume that we are analyzing a family of distributed-memory MIMD multiprocessors.
2) We assume that all processors in this family are identical, and that all processors have the same amount of local memory. We will refer to the number of floating point numbers that can be stored in this local memory by M.
3) For each example algorithm, we assume an interconnection topology that is natural for the algorithm. The underlying assumption is that these interconnection topologies can be embedded in the interconnection topologies of the multiprocessors in the family. In all examples, it will be sufficient if the interconnection topologies of the multiprocessors are binary hypercubes [21].
4) We assume that the time required to send k floating point numbers between neighboring processors can be described by the expression

$$t = \alpha + k \cdot \beta. \tag{4}$$

α is the startup time required to send any message and β is the incremental transmission time per floating point number [4], [7].

The following specific assumptions permit us to calculate the scaled speedup curves.

a) We assume that all floating point numbers have 8 bytes.

b) We assume that .45 Megabytes of local memory are available for program data storage per processor. Thus, there is enough memory to store 56,250 floating point numbers.

c) We assume that binary floating point addition, subtraction, multiplication, and division instructions all take approximately 8 μsec to execute in the (compute-intensive) type of programs that we will be analyzing. This is equivalent to a rate of .125 Megaflops for each processor. We will refer to the execution times of these operations by $f_{(+)}$, $f_{(-)}$, $f_{(*)}$, and $f_{(/)}$ respectively.

d) We assume that the communication startup time α is 376 μsec, and that the incremental transmission time β is 24 μsec.

e) We assume that computation and communication on the same processor are not overlapped. For example, if, during the execution of an algorithm, an addition operation is executed and a floating point number is sent to a neighboring processor, then the time required to execute these two operations is[3]

$$f_{(+)} + (\alpha + \beta). \tag{5}$$

These specific assumptions are not important to the analysis, but some assumptions are necessary to complete the analysis.

The model described above (assumptions 1) - 4) and a) - e)) allows us to identify when the bounds on performance established in [23] and [24] begin to affect actual algorithms. Different assumptions will change the analysis, and changes in multiprocessor technology will alter some of the conclusions drawn here. But the qualitative behavior indicated by the theory is independent of the architectural parameters, and the issue is only when the intrinsic limitations on the parallelism begin to affect performance.

3. Hyperbolic examples.

3.1. Two space dimensions. Consider the following hyperbolic equation in two space dimensions with periodic boundary conditions and a constant forcing function,

$$\frac{\partial^2}{\partial t^2}u(x,y,t) + \kappa \cdot \frac{\partial}{\partial t}u(x,y,t) - \left(\frac{\partial^2}{\partial x^2}u(x,y,t) + \frac{\partial^2}{\partial y^2}u(x,y,t)\right) = C \tag{6}$$

$$\text{for} \quad (x,y,t) \in [0,1] \times [0,1] \times [0,1],$$

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

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

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

[3] If communication and computation can be overlapped, then the execution times reported in the rest of this paper would, at most, be halved, and the speedups doubled. For example, the time required to add two floating point numbers together and send a floating point number to a neighboring processor is always bounded from below by $\max\{f_{(+)}, \alpha + \beta\}$, which is never less than half of the value of (5). Permitting overlap changes our conclusions very little.

where $0 < \kappa \ll 1$ and time is in units of seconds. Assume that we want the solution on a uniform mesh at time $T = 1$,

$$\{ (i\Delta s, j\Delta s, 1) \mid i,j \in \{1,\cdots,N_s\} \}, \tag{7}$$

where Δs is the distance between consecutive locations in each spatial coordinate direction and N_s^2 is the total number of locations, $N_s^2 = (1/\Delta s)^2$. Assume that values of the data functions h and g are available on the mesh $\{ (i\Delta s, j\Delta s) \mid i,j \in \{1,\cdots,N_s\} \}$.

We approximate $u(x,y,t)$ on the mesh at time $T = 1$ by timestepping using the standard second-order centered finite-difference formula,

$$\begin{aligned} \tilde{u}_{i,j}^{k+1} = {} & \left(\frac{4}{2+\kappa\cdot\Delta t}\right)\cdot\left(1-2\cdot\left(\frac{\Delta t}{\Delta s}\right)^2\right)\cdot\tilde{u}_{i,j}^{k} - \left(\frac{2-\kappa\cdot\Delta t}{2+\kappa\cdot\Delta t}\right)\cdot\tilde{u}_{i,j}^{k-1} \\ & + \left(\frac{2\cdot(\Delta t)^2}{(2+\kappa\cdot\Delta t)\cdot(\Delta s)^2}\right)\cdot\left(\tilde{u}_{i+1,j}^{k}+\tilde{u}_{i-1,j}^{k}+\tilde{u}_{i,j+1}^{k}+\tilde{u}_{i,j-1}^{k}\right) \\ & + \left(\frac{2\cdot(\Delta t)^2}{2+\kappa\cdot\Delta t}\right)\cdot C \qquad \forall i,j \in \{1,\cdots,N_s\} \quad \forall k \in \{1,\cdots,N_t\}, \end{aligned} \tag{8}$$

where Δt is the length of the timestep and $N_t = 1/\Delta t$. $\tilde{u}_{i,j}^k$ is an approximation to u at the location $(i\Delta s, j\Delta s, k\Delta t)$. By periodicity, $\tilde{u}_{N_s,j}^k = \tilde{u}_{0,j}^k$ for all j and k, and $\tilde{u}_{i,N_s}^k = \tilde{u}_{i,0}^k$ for all i and k. This scheme requires approximations to the solution at times $(k-1)\Delta t$ and $k\Delta t$ in order to calculate an approximation to the solution at time $(k+1)\Delta t$. To start the process, we use a slight modification of (8) to calculate the solution at time Δt from the initial data [15, pp. 565-566]. If we precompute the constant factors, then the serial complexity of the calculation is approximately

$$\left(6\cdot f_{(+)} + 3\cdot f_{(*)}\right)\cdot N_s^2\cdot N_t. \tag{9}$$

The stability condition for this algorithm is $\Delta t/\Delta s \le 1/\sqrt{2} - \epsilon$ for some positive $\epsilon < 1/\sqrt{2}$ [20].

The computation of the approximation at time $(k+1)\Delta t$ from the approximation at times $(k-1)\Delta t$ and $k\Delta t$ is easily parallelized. Assume that the multiprocessor has P processors and can be configured as a toroidal mesh with four nearest neighbors per processor and two communication channels between neighboring processors (allowing duplex communications). If $N_s/\sqrt{P}$ is an integer, then partition the square $[0,1]\times[0,1]$ into P equal subsquares, map the subsquares and data onto the processors in such a way as to preserve the topology of the problem domain, and assign the calculation of the solution at the locations in each subsquare to the corresponding processor. Then each processor needs to receive only $N_s/\sqrt{P}$ floating point numbers from each of its neighbors in order to finish its calculation of the next timestep.[4] Since computation and communication are not overlapped, the total execution time for this parallel implementation is

$$\left(\left(6\cdot f_{(+)} + 3\cdot f_{(*)}\right)\cdot\frac{N_s^2}{P} + 4\cdot\left(\alpha+\beta\cdot\frac{N_s}{\sqrt{P}}\right)\right)\cdot N_t. \tag{10}$$

[4] Mappings based on partitioning the problem domain into hexagons or rectangular strips have also been used to generate parallel implementations [19]. For the assumed values of α and β, the mapping described here is better than either of those alternatives when $P \ge 16$ and $N_s \ge 64$.

A similar implementation can be described if $N_s/\sqrt{P}$ is not an integer, and expression (10) is then a lower bound on the execution time. Note that when more than N_s^2 processors are used to parallelize this algorithm, the execution time begins to increase. The decrease in computation time is more than offset by the increase in the time spent in interprocessor communication. This fact follows from the number of messages that must be sent (and so is true for any interconnection topology) and from the assumed speeds of communication and computation. In consequence, we will not consider using more than N_s^2 processors.

For a fixed number of processors, the execution time of this parallel algorithm is a function of only Δt and Δs. The optimal choice of Δt and Δs either minimizes the approximation error subject to a bound on the execution time or minimizes the execution time subject to a bound on the approximation error, depending on which constraint is more binding. We will consider the first definition initially. For a stable algorithm, minimizing the truncation error of the discretization [20] is a good heuristic for minimizing the approximation error. Without a priori knowledge of the solution function, the best approximation to the truncation error has the form

$$C \cdot \left((1 + 4 \cdot \kappa) \cdot (\Delta t)^2 + 2 \cdot (\Delta s)^2\right). \tag{11}$$

Thus, the optimal values of Δt and Δs are calculated by minimizing (11) subject to the stability condition and the bound on the execution time, and they satisfy the condition $\Delta t/\Delta s \approx 1/\sqrt{2}$. It is straightforward to show that this condition is also satisfied by the choice of Δt and Δs that minimizes the execution time with respect to a bound on the approximation error. For the rest of the analysis, we will assume that $\Delta t/\Delta s = 1/\sqrt{2}$, with the understanding that this is a good approximation to what would be used in practice. Equivalently, we are assuming that $N_t = \sqrt{2} \cdot N_s$. *Thus, the number of timesteps N_t is an unbounded monotonic function of the size of the spatial grid N_s^2. By* (9) *and* (10), *this implies that the execution time will grow unboundedly as a function of the serial complexity, regardless of the number of processors used.*

3.1.1. Fixed-size speedup curves. To begin the calculation for a given timestep, enough memory must be available to hold the approximate solution at the two previous timesteps. Thus, the entire computation can proceed on one processor as long as a little more than $2 \cdot N_s^2$ floating point numbers can be stored, and the maximum value of N_s for a serial implementation of the algorithm is approximately $\sqrt{M/2}$. By our assumptions, this corresponds to $N_s \approx 167$, or $\Delta s \approx .006$. This assumes that the solution is required only at time $T = 1$ and that the size of the memory is the active constraint in determining the size of the problem. If, instead, the solution of the problem must proceed at least as fast as real time, then the execution time must be no greater than one second when approximating the solution at time $T = 1$. In this case the maximum value of N_s for a serial implementation of the algorithm is

$$N_s \approx \left(\frac{\sqrt{2}}{6 \cdot f_{(+)} + 3 \cdot f_{(*)}}\right)^{1/3} \approx 21, \tag{12}$$

and $\Delta s \approx .048$.

Figure 1 contains the graphs of the speedup curves for these two fixed-size examples. As expected, both curves level off long before the maximum number of processors N_s^2 is reached. The time-constrained example has a maximum speedup of approximately 19, using 441 processors. The efficiency is less than 50 percent when

$P > 15$. In contrast, the memory-constrained example has a maximum speedup of approximately 1,201, using 27,889 processors. The efficiency falls below 50 percent only when $P > 998$.

3.1.2. Memory-constrained scaled speedup curve. Next, we let the size of the problem grow to fill the available memory as the number of processors increases. Since each additional processor adds the capacity to store M additional floating point numbers, the bound on N_s is now

$$N_s \approx \sqrt{\frac{M \cdot P}{2}} = \sqrt{28,125 \cdot P}. \tag{13}$$

We will refer to this as the memory-constrained model. This function is graphed in Fig. 2.

Figure 3 contains the graph of the memory-constrained scaled speedup curve. This function grows linearly as a function of P with a slope near one,

$$S = \frac{(6 \cdot f_{(+)} + 3 \cdot f_{(*)}) \cdot M}{(6 \cdot f_{(+)} + 3 \cdot f_{(*)}) \cdot M + 8 \cdot (\alpha + \beta \cdot \sqrt{M/2})} \cdot P \approx .9914 \cdot P. \tag{14}$$

Thus, very good speedup and efficiency are maintained for any number of processors. This analysis does not indicate any limit to the number of processors that can be used effectively, but the execution time for the memory-constrained model is

(15)

$$\sqrt{M} \cdot \left((6f_{(+)} + 3f_{(*)}) \cdot \frac{M}{2} + 4 \left(\alpha + \beta \cdot \sqrt{\frac{M}{2}} \right) \right) \cdot \sqrt{P} \approx 484\sqrt{P} \text{ seconds,}$$

and any time constraint will eventually be exceeded as P increases. Figure 4 contains the graph of the execution time of the memory-constrained model as a function of P. The execution time is approximately 7.9 minutes when one processor is used, and it increases to over four hours by the time 1,000 processors are used. If 1,000,000 processors are used, then the execution time is approximately 134 hours.

3.1.3. Time-constrained scaled speedup curve. Finally, we let the size of the problem grow to satisfy the real-time bound on the execution time. To satisfy the time constraint for the P processor parallel implementation, we need

$$\left((6 \cdot f_{(+)} + 3 \cdot f_{(*)}) \cdot \frac{N_s^2}{P} + 4 \cdot \left(\alpha + \beta \cdot \frac{N_s}{\sqrt{P}} \right) \right) \cdot \sqrt{2} \cdot N_s \leq 1. \tag{16}$$

This bound on N_s is graphed in Fig. 2. We will refer to this as the time-constrained model. Since P is constrained to be less than or equal to N_s^2, there is a largest problem that can be solved, regardless of the number of available processors. For this problem, N_s cannot be larger than 422. Note that this is larger than either of the two fixed-size examples, so we expect more promising results for the time-constrained model.

Figure 3 contains the time-constrained scaled speedup curve for this problem and the speedup curve for the fixed-size problem $N_s = 422$. The maximum speedup is approximately 7,669, using 178,084 processors. The efficiency does not fall below 50 percent until $P > 1,108$. The time-constrained scaled speedup curve has many of the same characteristics as fixed-size speedup curves. There is an upper bound on the maximum number of processors that can be used, and most of the speedup

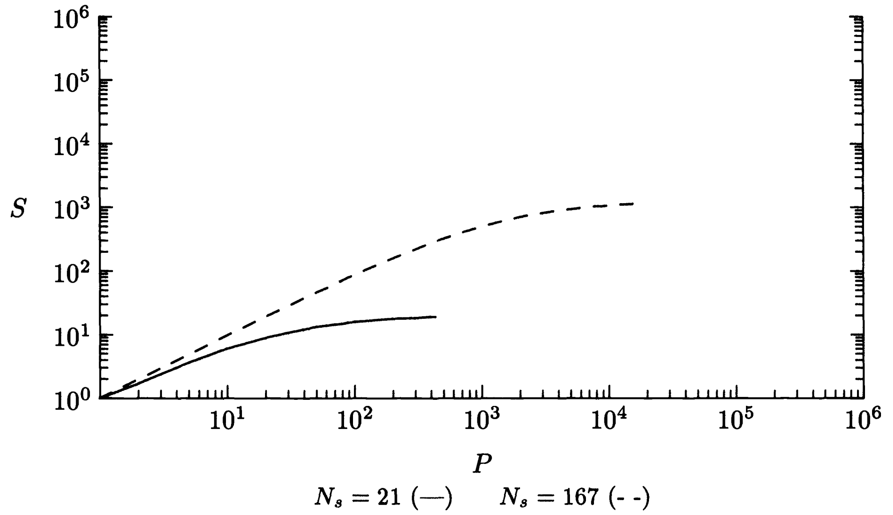


FIG. 1. *Speedup curves for fixed-size examples of the two-dimensional hyperbolic problem.*

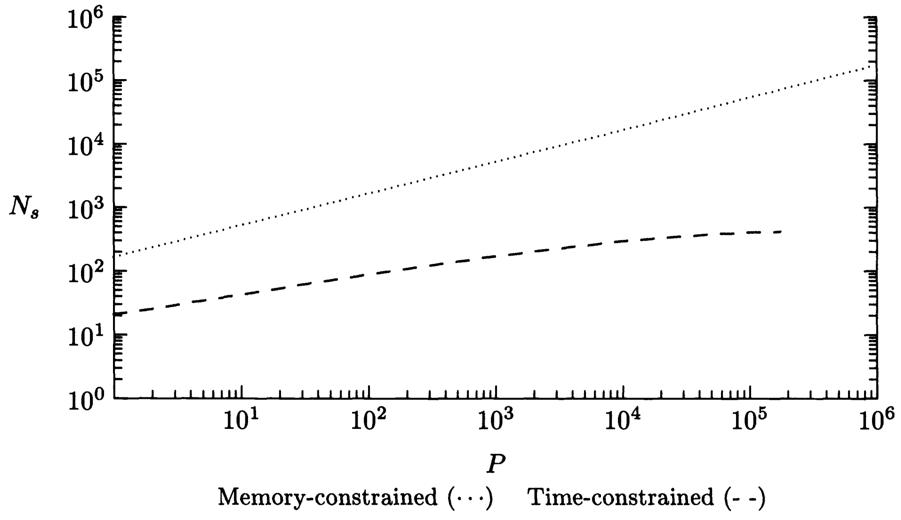
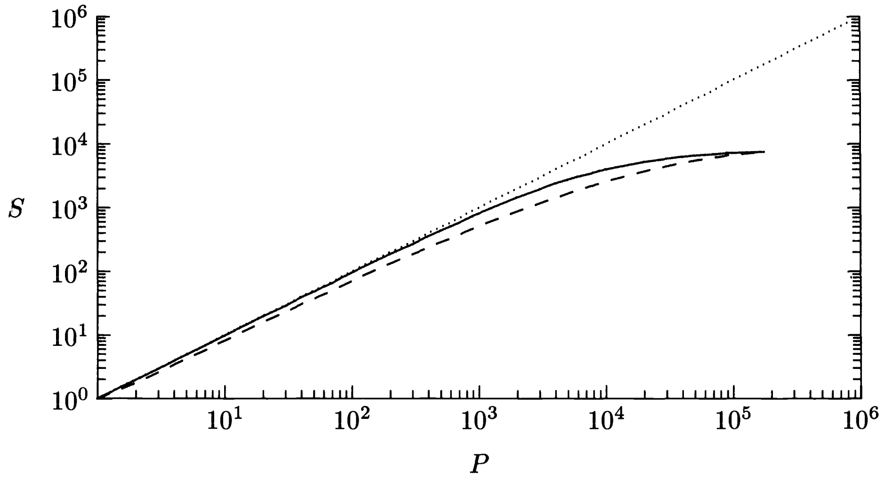
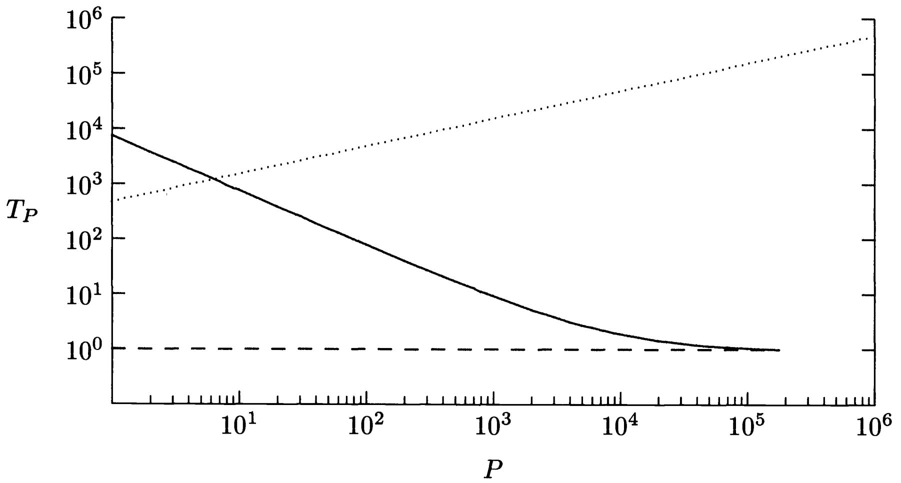


FIG. 2. *Bounds on scaled problem sizes for the two-dimensional hyperbolic problem.*



Memory-constrained (· · ·) Time-constrained (- -) $N_s = 422$ (—)

FIG. 3. *Scaled speedup curves for the two-dimensional hyperbolic problem.*



Memory-constrained (· · ·) Time-constrained (- -) $N_s = 422$ (—)

FIG. 4. *Execution time in seconds for scaled speedup models versus number of processors for the two-dimensional hyperbolic problem.*

is achieved when using significantly fewer processors. Moreover, the speedup curve for the maximum size problem $N_s = 422$ is a reasonable approximation to the time-constrained scaled speedup curve.

In conclusion, unlike for the memory-constrained model, an unlimited number of processors cannot be used. But tens of thousands of processors can still be utilized effectively. If 50 percent utility is required, then a maximum of 1,108 processors can be used, but half of the maximum possible speedup is not achieved until approximately 21,000 processors are used.

3.2. Generalizations.

3.2.1. Other parallel implementations. The parallel implementation used in §3.1 is not optimal, although it does appear to be fairly good. But, (8) specifies a partial order constraining when values can be calculated in any parallel implementation. In particular, it is clear that at least N_t parallel "steps" are required to calculate the solution at time $T = 1$ using any parallel implementation. Thus, the execution time must grow at least linearly in N_t, which, by the stability condition, must grow at least as fast as $\sqrt{2} \cdot N_s$. Because of this, the behavior of the time-constrained model will be qualitatively the same for any reasonable parallel implementation.

Note that the choice of the parallel implementation is closely tied to the interconnection topology. For the given algorithm, there is little to be gained by increasing the connectivity, but decreasing the connectivity can change the decision of how best to partition the domain. For example, if the two-dimensional example is to be solved on a ring topology, then the natural parallel implementation uses a strip partition of the spatial domain (as long as $N_s > P$). This increases the communication cost for large problem sizes and the time-constrained model becomes more pessimistic, but the results do not change qualitatively.

3.2.2. Other problems. The analysis described in §3.1 can be extended easily to handle other PDEs that are discretized using traditional explicit finite-difference and finite-element schemes. Nothing we have done is particularly sensitive to either the boundary conditions, if they are local, or the shape of the domain. Using different differential operators, or systems of differential operators, will change the number of arithmetic operations, the coefficient of β, and (less likely) the coefficient of α in expression (10) by some fixed amount. Thus, the quantitative results will be scaled, but the qualitative results are unchanged as long as N_t is a similar increasing function of the size of the problem.

3.2.3. Stability. The stability condition is an active constraint in the minimization problem that determines the optimal Δs and Δt for the algorithm described in §3.1, but there would be little change in the analysis if the algorithm were unconditionally stable. For example, assume that the algorithm is stable even when the ratio $\Delta t/\Delta s$ is arbitrarily large. Now we can use an arbitrarily large number of processors even when the execution time is bounded if we keep Δt fixed and make Δs sufficiently small. But, decreasing Δs for a fixed Δt at most decreases the truncation error to one third of its previous value, and the serial complexity for this choice of Δt and Δs becomes arbitrarily large. The same decrease in the truncation error can be achieved by decreasing Δt and Δs equally by a factor of $1/\sqrt{3}$, at the cost of only increasing the serial complexity by a factor of $3 \cdot \sqrt{3}$. If the speedup is measured relative to the execution time of the serial algorithm that achieves the same error bound (using the optimal choice of Δt and Δs), then the time-constrained scaled speedup is at most increased by a factor of $3 \cdot \sqrt{3}$ by allowing $\Delta s \to 0$ and $P \to \infty$, and at the cost of the efficiency going to zero. Thus, the practical limits on the number of processors that can be used do not change significantly.

3.2.4. Other space dimensions. It is simple to extend the analysis in §3.1 to second-order centered finite-difference schemes for the corresponding scalar hyperbolic PDEs in other space dimensions. For a general d-dimensional cube, the serial algorithm has a complexity of

$$(17) \qquad \left(2 \cdot (d+1) \cdot f_{(+)} + 3 \cdot f_{(*)}\right) \cdot N_s^d \cdot N_t.$$

Assume that we are using a multiprocessor whose interconnection topology is a d-dimensional mesh supplemented by links between corresponding processors in opposing faces of the mesh. If we partition the spatial domain into d-dimensional subcubes and map the blocks onto processors in the natural way, then the resulting parallel algorithm has an execution time of

$$(18) \quad \left(\left(2 \cdot (d+1) \cdot f_{(+)} + 3 \cdot f_{(*)}\right) \cdot \frac{N_s^d}{P} + 2 \cdot d \cdot \left(\alpha + \beta \cdot \left(\frac{N_s}{P^{1/d}}\right)^{d-1}\right)\right) \cdot N_t.$$

The stability condition for this algorithm is $\Delta t/\Delta s \leq 1/\sqrt{d} - \epsilon$ for some positive $\epsilon < 1/\sqrt{d}$, and the optimal choice of Δt and Δs satisfies $\Delta t/\Delta s \approx 1/\sqrt{d}$. Thus, $N_t \approx \sqrt{d} \cdot N_s$. The memory-constrained model assumes that

$$(19) \qquad N_s \approx \left(\frac{M \cdot P}{2}\right)^{1/d}.$$

The memory-constrained scaled speedup is

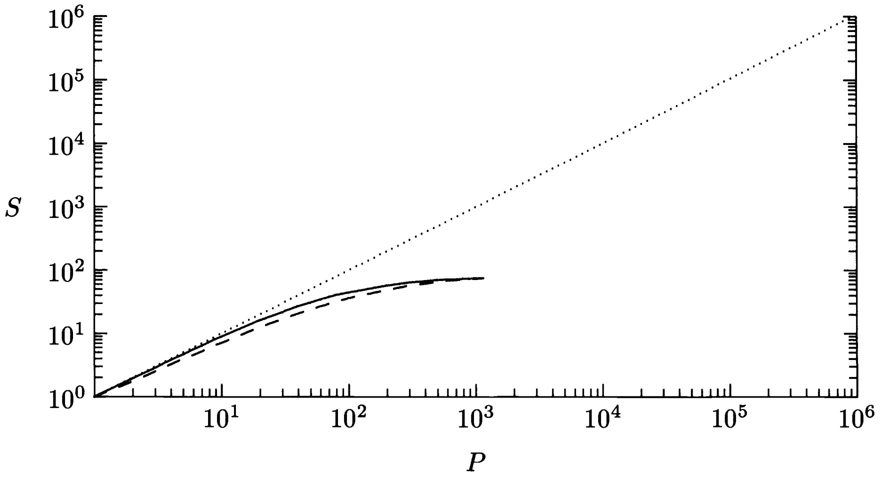
$$(20) \quad S = \frac{(2 \cdot (d+1) \cdot f_{(+)} + 3 \cdot f_{(*)}) \cdot M}{(2 \cdot (d+1) \cdot f_{(+)} + 3 \cdot f_{(*)}) \cdot M + 4 \cdot d \cdot \left(\alpha + \beta \cdot (M/2)^{(d-1)/d}\right)} \cdot P$$

and the execution time of the memory-constrained model is

$$(21)$$
$$\sqrt{d} \cdot \left(\left(2(d+1) \cdot f_{(+)} + 3f_{(*)}\right) \cdot \frac{M}{2} + 2d \cdot \left(\alpha + \beta \cdot \left(\frac{M}{2}\right)^{(d-1)/d}\right)\right) \cdot \left(\frac{M \cdot P}{2}\right)^{1/d}.$$

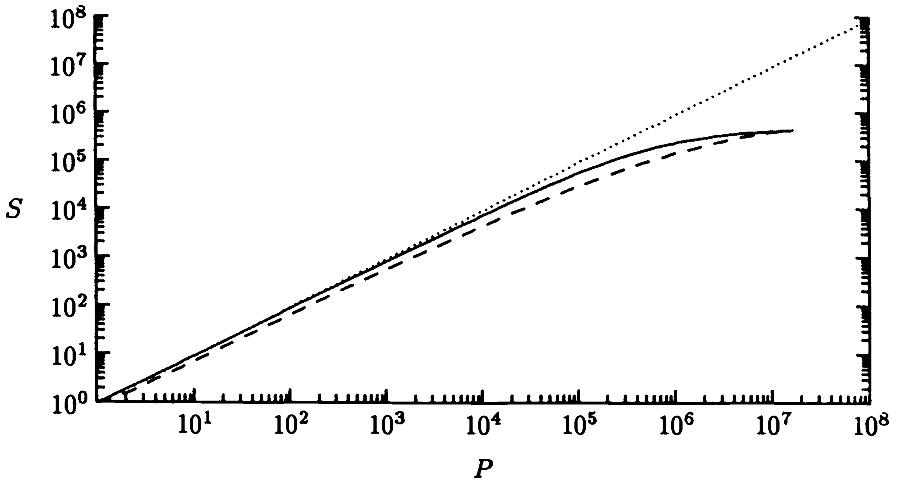
Thus, the memory-constrained scaled speedup is always a linear function of P, and the memory-constrained model always indicates that any number of processors can be used. But, the execution time of this model increases linearly in $P^{1/d}$, and any time constraint will eventually be violated. The larger d is, the slower this growth is, and the more optimistic we expect the time-constrained model to be. For example, Figs. 5 and 6 contain the memory-constrained and time-constrained scaled speedup curves for the cases $d = 1$ and $d = 3$, respectively.

For $d = 1$, time-constrained model is more pessimistic than before. The maximum number of processors that can be used is 1168, which provides a speedup of 76, and the efficiency falls below 50 percent when $P > 43$. In contrast, for $d = 3$ the maximum number of processors that can be used is 12,487,168, which provides a speedup of approximately 441,668, and the efficiency only falls below 50 percent when $P > 7,000$.



Memory-constrained ($\cdots$) Time-constrained (- -) $N_s = 1168$ (—)

FIG. 5. *Scaled speedup curves for the one-dimensional hyperbolic problem.*



Memory-constrained ($\cdots$) Time-constrained (- -) $N_s = 232$ (—)

FIG. 6. *Scaled speedup curves for the three-dimensional hyperbolic problem.*

4. Elliptic examples.

4.1. One space dimension. Consider the following elliptic equation in one space dimension with Dirichlet boundary conditions,

$$\frac{\partial^2}{\partial x^2}u(x) \;-\; \alpha(x)\cdot u(x) \;=\; g(x) \quad \text{for } \; x \in [0,1], \tag{22}$$

$$u(0) \;=\; 0 \;=\; u(1),$$

where $\alpha(x)$ is a nonnegative function. Assume that we want the solution on a uniform mesh,

$$\{\, j\Delta s \,|\, j \in \{1,\cdots,N_s\}\,\}, \tag{23}$$

where Δs is the distance between consecutive locations and N_s is the total number of locations, $N_s = (1/\Delta s \,-\, 1)$. Assume that values of the functions $\alpha(x)$ and $g(x)$ are also available on this mesh.

We approximate $u(x)$ by replacing the differential equation (22) by the standard second-order centered finite-difference scheme and solving the resulting coupled system of linear equations:

$$\begin{aligned} \tilde{u}_2 - \left(2+(\Delta s)^2\cdot\alpha_1\right)\cdot\tilde{u}_1 &= (\Delta s)^2\cdot g_1, \\ \tilde{u}_{j+1} - \left(2+(\Delta s)^2\cdot\alpha_j\right)\cdot\tilde{u}_j + \tilde{u}_{j-1} &= (\Delta s)^2\cdot g_j \quad \forall j \in \{2,\cdots,N_s-1\}, \\ -\left(2+(\Delta s)^2\cdot\alpha_{N_s}\right)\cdot\tilde{u}_{N_s} + \tilde{u}_{N_s-1} &= (\Delta s)^2\cdot g_{N_s}. \end{aligned} \tag{24}$$

$\tilde{u}_j$ is an approximation to u at the location $j\Delta s$, $\alpha_j = \alpha(j\Delta s)$, and $g_j = g(j\Delta s)$. We will also refer to this system by the matrix equation $A\bar{u} = \bar{g}$, where A is a symmetric positive-definite tridiagonal $N_s \times N_s$ matrix, $\bar{u}$ is the vector of approximate solution values, and $\bar{g}$ is the vector of data.

There are numerous techniques for solving this matrix equation. If a point iterative method is used [10], [25], then the analysis is very similar to that described in §3. Each step of the iteration involves a weighted average of values associated with neighboring mesh locations, and the number of iterations is approximately a linear function of N_s^α for some positive α. Many of the standard serial and parallel algorithms for this problem have a structure similar to that outlined in Fig. 7. When $N_s + 1$ is an integer power of two this description applies to multigrid[5] [3], Gaussian elimination using the nested dissection ordering of the rows and columns of A [8], and cyclic reduction [13]. The approach can be modified to work even if N_s is not a power of two. We will use the cyclic reduction algorithm for our analysis. The serial complexity of the cyclic reduction algorithm is almost three times as large as that of the best known serial algorithm for this problem [13], but the parallel implementation described below is a competitive parallel algorithm [14]. A rough comparison of this parallel algorithm with the best serial algorithm can be calculated by dividing all speedup values in the rest of this section by three.

[5] Since multigrid only approximately reduces the system in step 1), steps 0) - 5) will need to be repeated a number of times until the process converges.

0) Let $N = N_s + 1$.
1) Given a tridiagonal system of $N-1$ equations representing $N-1$ mesh locations, generate a new tridiagonal system of $N/2-1$ equations whose solution is the even numbered values of the solution vector of the larger system. Set $N = N/2$.
   There is now one equation for every location on a coarser mesh. We will refer to this as the *active* mesh during this stage of the algorithm. Each new equation is a weighted sum of the original equation corresponding to that mesh location and the equations corresponding to the mesh locations on either side of it in the original fine mesh.
2) Repeat step 1) until the system is reduced to a single equation, saving all of the intermediate systems.
3) Solve the one equation and set $N = 2$.
4) Use the solution of the system of size $N/2-1$ to solve for the remaining unknowns in the tridiagonal system of size $N-1$. Set $N = 2 \cdot N$. We will refer to the corresponding mesh locations as being active during this part of the algorithm.
5) Repeat step 4) until the original problem is solved.

FIG. 7. *Outline of the cyclic reduction algorithm for the one-dimensional elliptic problem.*

The details of cyclic reduction can be found in Hockney and Jesshope [13, pp. 286-298]. For now, assume that $N \equiv N_s + 1$ is an integer power of two. Step 1) requires 4 additions, 6 multiplications, and 2 divisions per new equation, and step 1) is executed $\log_2 N - 1$ times. Step 3) requires 1 division, and is executed once. Step 4) requires 2 additions, 2 multiplications, and 1 division per new solution value, and step 4) is executed $\log_2 N - 1$ times. Thus, the serial complexity is

$$(25) \quad \left(6f_{(+)} + 8f_{(*)} + 3f_{(/)}\right) \cdot (N-1) - \left(4f_{(+)} + 6f_{(*)} + 2f_{(/)}\right) \cdot \log_2 N + f_{(/)} \ .$$

For simplicity, we will assume that the same complexity holds when N is not an integer power of two. This algorithm requires storage for approximately $7 \cdot N_s$ floating point values.

Most of the work in steps 1) and 4) of the algorithm can be done in parallel. Assume that the multiprocessor has P processors and can be configured as a binary hypercube with two communication channels between neighboring processors. Partition the interval $[0,1]$ into P equal subintervals, map neighboring subintervals to neighboring processors, and assign the calculation of the solution at the locations in each subinterval to the corresponding processor. As long as there is at least one active mesh location associated with each processor, each processor needs to receive only four floating point numbers from each of two neighbors to finish the current iteration of step 1), and one floating point number from each of two neighbors to finish the current iteration of step 4). During these stages of the algorithm only a linear array interconnection topology is being used by the parallel implementation. For some iterations of steps 1) and 4) the active mesh is too coarse for all processors to have active mesh locations, and some processors will be idle. But, if some care is taken when mapping the subintervals to the processors, the processors that need to communicate will only be a distance of two apart [14]. That is, the information must pass through only one intermediate processor. This uses the entire hypercube interconnection network.

If both N and P are integer powers of two and $N \geq 2 \cdot P$, then the execution time of this parallel implementation is

(26)

$$\left(6 \cdot f_{(+)} + 8 \cdot f_{(*)} + 3 \cdot f_{(/)}\right) \cdot \left(\frac{N-P}{P} + \log_2 P\right) + \alpha \cdot (4 \cdot \log_2 N + 4 \cdot \log_2 P - 8)$$
$$+ \beta \cdot (10 \cdot \log_2 N + 10 \cdot \log_2 P - 26) - \left(4 \cdot f_{(+)} + 5 \cdot f_{(*)} + f_{(/)}\right) .$$

This is a lower bound on the execution time otherwise. *Thus, as in the examples in §3, the execution time of this parallel algorithm will grow unboundedly as a function of the serial complexity, regardless of the number of processors. But now the lower bound on the execution time is a logarithmic function of the serial complexity, and the growth is much slower.* For this implementation there is no advantage to having more than $N/2$ processors. While more can be used, the execution time begins to grow due to the additional time spent in interprocessor communication. Each processor needs to hold at least $7 \cdot N_s/P$ floating point values, but at least one needs to hold $7 \cdot N_s/P + 4 \cdot \log_2(N_s/P)$ floating point values.

4.1.1. Fixed-size speedup curves. The algorithm can be executed on one processor as long as approximately $7 \cdot N_s$ floating point numbers can be stored. If the size of the memory is the only limitation on the size of the problem, then the maximum value of N_s in a serial implementation of the algorithm is approximately $M/7$. By our assumptions, this corresponds to $N_s \approx 8035$, or $\Delta s \approx .00012$. If the execution time of the problem must be less than one second, then the maximum value of N_s in a serial implementation is 7,362, or $\Delta s \approx .00014$. This time constraint is arbitrary for this problem since there is no corresponding real time. But some bound will hold in practice, and this one is appropriate given the earlier analyses. Unlike the example in §3.1, the effect of the memory constraint is comparable to the effect of the time constraint. For the rest of this section, we will consider only the fixed-size problem $N_s = 8035$.

Figure 8 contains the graph of the speedup curve for this fixed-size example. While over 4,000 processors can be used, the speedup begins decreasing when more than 403 processors are used, due to the presence of the terms that grow as a function of P. The maximum speedup is approximately 29, using 403 processors. The efficiency is less than 50 percent when $P > 38$.

4.1.2. Memory-constrained scaled speedup curve. If the size of the problem grows to fill the available memory, then N_s satisfies

$$7 \cdot \frac{N_s}{P} + 4 \cdot \log_2\left(\frac{N_s}{P}\right) \leq M \tag{27}$$

when $P > 1$. A good approximation to the maximum value that satisfies this inequality is $N_s = MP/7$, or $N_s \approx 8035 \cdot P$ by our assumptions. We will refer to this as the memory-constrained model. For large M, a good approximation to the memory-constrained scaled speedup is

$$S \approx \left(\frac{C \cdot M}{C \cdot M + 7 \cdot (C + 8 \cdot \alpha + 20 \cdot \beta) \cdot \log_2 P}\right) \cdot P, \tag{28}$$

where $C = \left(6 \cdot f_{(+)} + 8 \cdot f_{(*)} + 3 \cdot f_{(/)}\right)$, and a good approximation to the execution time of the memory-constrained model is

$$C \cdot \frac{M}{7} + (C + 8 \cdot \alpha + 20 \cdot \beta) \cdot \log_2 P. \tag{29}$$

By our assumptions, these approximations correspond to a scaled speedup of

$$S \approx \left(\frac{301.5}{301.5 + \log_2 P}\right) \cdot P \tag{30}$$

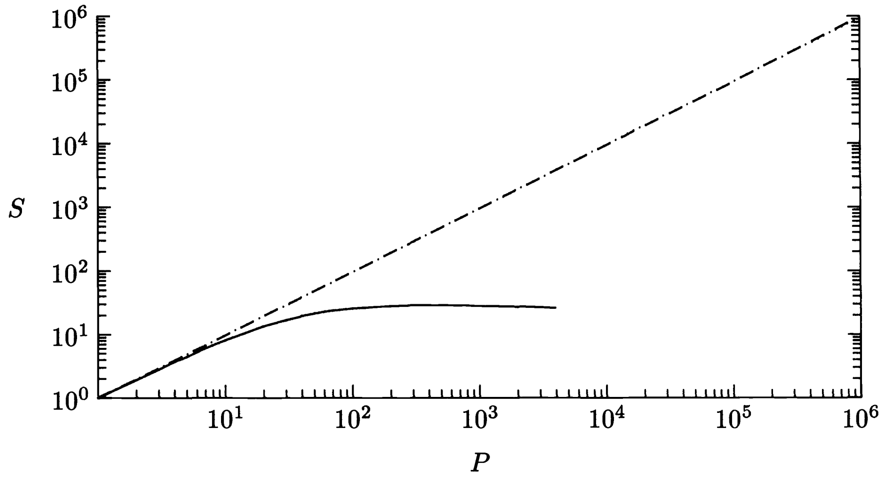


FIG. 8. *Scaled speedup curves for the one-dimensional elliptic problem.*

and an execution time of approximately $(1.09 + .0036 \cdot \log_2 P)$ seconds.

Figure 8 contains the graph of the memory-constrained scaled speedup curve. The speedup appears to grow linearly as a function of P with a slope near one, in agreement with (30) for the numbers of processors considered. Thus, like all of the previous memory-constrained examples, very good speedup and efficiency are maintained for any practical number of processors. Figure 9 contains the graph of the execution time of the memory-constrained model as a function of P. Note that, unlike the previous graphs, this is a log-linear graph with a small linear scale. For this problem the execution time hardly increases at all. The execution time is approximately 1.09 seconds when one processor is used, and increases to only 1.18 seconds when 1,000,000 processors are used. Thus, the implications of the memory-constrained model are both optimistic and believable.

4.1.3. Time-constrained scaled speedup curve. To satisfy a one second bound on the execution time, N_s must satisfy

(31)
$$\left(6 \cdot f_{(+)} + 8 \cdot f_{(*)} + 3 \cdot f_{(/)}\right) \cdot \left(\frac{N-P}{P} + \log_2 P\right) + \alpha \cdot (4 \cdot \log_2 N + 4 \cdot \log_2 P - 8)$$
$$+ \beta \cdot (10 \cdot \log_2 N + 10 \cdot \log_2 P - 26) - \left(4 \cdot f_{(+)} + 5 \cdot f_{(*)} + f_{(/)}\right) \leq 1$$

when $P > 1$. We will refer to this bound on N_s as the time-constrained model. While the model has a maximum size problem that can be solved, the size of the time-constrained model grows almost as fast as the memory-constrained model for all numbers of processors examined. Figure 8 contains the time-constrained scaled speedup curve for this problem. The time-constrained scaled speedup is just as optimistic as the memory-constrained scaled speedup, indicating no practical limits on the number of processors that can be used. This is in distinct contrast with the

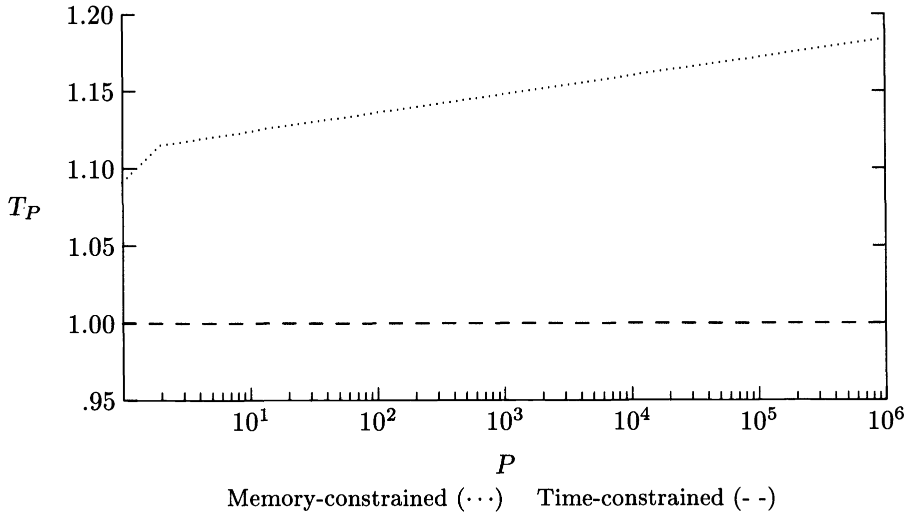


FIG. 9. *Execution time in seconds for scaled speedup models versus number of processors for the one-dimensional elliptic problem.*

fixed-size speedup curve, which was very pessimistic. *The decision as to whether a massively parallel processor is appropriate for this problem is very sensitive to the degree to which a scaled analysis is appropriate.*

4.2. Generalizations.

4.2.1. Other interconnection topologies. Increasing the connectivity of the interconnection topology can decrease the execution time of the algorithm in §4.1. For example, if the interconnection topology is fully connected, then all interprocessor communications is between neighboring processors, and two of the three $\log_2 P$ terms in (26) disappear. But, this at most halves the time spent in interprocessor communication, and the results of the analysis change very little.

In contrast, decreasing the connectivity of the interconnection topology changes the results of §4.1 significantly. To indicate the degree to which the conclusions can change, assume that the multiprocessor family can support only a one-dimensional mesh interconnection topology. In this case processors must send messages progressively further as the active mesh becomes coarser in steps 1) and 4) of the algorithm. The execution time of this modified parallel implementation is

(32)
$$\left(6f_{(+)} + 8f_{(*)} + 3f_{(/)}\right) \cdot \left(\frac{N-P}{P} + \log_2 P\right) + \alpha \cdot (4\log_2 N - 4\log_2 P + 4P - 6)$$
$$+ \beta \cdot (10\log_2 N - 10\log_2 P + 10P - 45) - \left(4f_{(+)} + 5f_{(*)} + f_{(/)}\right).$$

This expression grows linearly as a function of P, and the fixed-size example is even more pessimistic than before. This is indicated by the speedup curve in Fig. 10. The speedup begins decreasing when $P > 26$. The maximum speedup is approximately 11, using 26 processors, and the efficiency is less than 50 percent when $P > 21$.

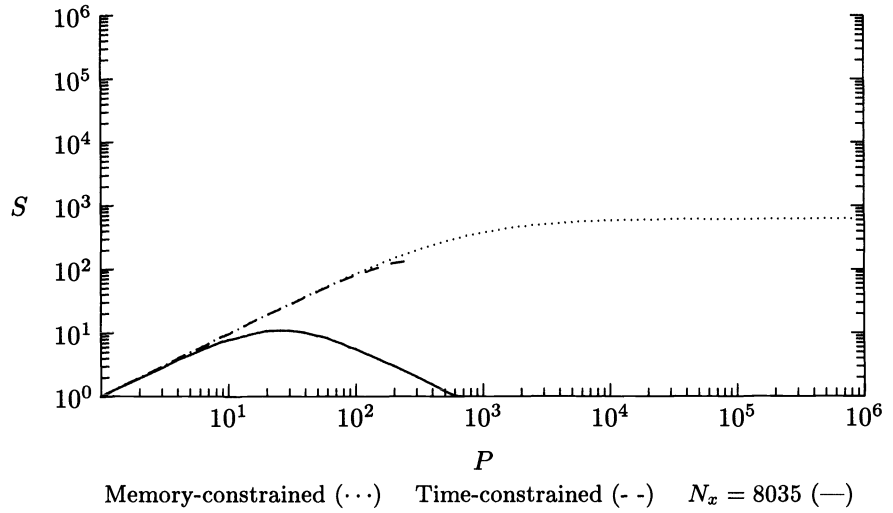


FIG. 10. *Scaled speedup curves for the one-dimensional elliptic problem when a one-dimensional mesh topology is used.*

Figure 10 also contains the graph of the memory-constrained scaled speedup. While there is still no upper bound on the number of processors that can be used, there is now an upper bound on the achievable speedup. For our assumptions, the maximum speedup is approximately 627 and the efficiency falls below 50 percent when $P > 614$. Even this limited amount of speedup comes at the cost of an increase in the execution time. The execution time for one processor is still approximately 1.09 seconds, but it increases to approximately 29 minutes when $P = 1{,}000{,}000$.

The graph of the time-constrained scaled speedup is also in Fig. 10. Since the expression for the execution time has a term that is linear in P, at some point an increase in P will decrease the size of the problem that can be solved and still satisfy the time constraint. This occurs when $P = 282$, and $N_s = 1{,}014{,}163$. This is a practical limit to the number of processors that can be used. The basic assumption behind the time-constrained scaled speedup analysis is that the goal is to solve larger problems, subject to the time constraint. For $P > 282$, the size of the problem can no longer increase, and there is no advantage to using more processors. The speedup is 138 when $P = 282$, and the efficiency is 49 percent.

In general, if a multiprocessor family is limited to a k-dimensional mesh interconnection topology, then the execution time will have a term that grows like $P^{1/k}$. Thus, a scaled speedup analysis indicates that the utility of a large number of processors will increase monotonically as a function of k, with the one-dimensional mesh and the hypercube topologies representing the limiting cases. Note that the fixed-size analysis always indicates that relatively few processors can be used, independent of the interconnection topology.

4.2.2. Higher space dimensions. An analysis similar to that of §4.1 follows for the generalizations of (22) and (24) to higher dimensions if a multigrid solver is used. For example, consider a single V-cycle of a multigrid algorithm that uses a point iterative relaxation scheme and local interpolation operators for projecting the solution between grids [22]. The serial complexity of this algorithm has the approximate form

$$C \cdot d \cdot N_s^d, \tag{33}$$

where d is the dimension. For a multiprocessor with a hypercube interconnection topology and a parallel implementation that preserves the topology of the problem domain, the execution time of the parallel algorithm has the approximate form

$$C_1 \cdot d \cdot \frac{N_s^d}{P} + C_2 \cdot d \cdot \left(\frac{N_s}{P^{1/d}}\right)^{d-1} + C_3 \cdot d \cdot \log_2 N_s + C_4 \cdot \log_2 P \tag{34}$$

for positive constants C_1, C_2, C_3, and C_4.

A multigrid algorithm will require multiple iterations to solve the problem, but the number of iterations required by a well-designed multigrid algorithm will be either independent of N_s or a very slowly growing function of N_s. Thus, it is reasonable to assume that (34) is the correct form for the execution time of the complete parallel algorithm. Just as in the one-dimensional example, this expression has a term that grows like $\log_2 P$ and a term that grows like $\log_2 N_s$. Thus, we expect a fixed-size example to be very pessimistic and the memory-constrained and time-constrained models to be very optimistic. Similarly, if the multiprocessor family is limited to a k-dimensional mesh interconnection topology, then the execution time will have a term that grows like $P^{1/k}$, and even the memory-constrained model may indicate a practical limit on the number of processors that can be used.

5. Conclusions.

The speedup curves for the current workload are often the most appropriate measures to use to decide how many processors to buy when buying a new multiprocessor or upgrading an old one. But, if the sizes of some of the problems are expected to grow, then scaled speedup curves can also be appropriate measures. Since the time-constrained scaled speedup curve and the memory-constrained scaled speedup curve are intrinsically different, it is important to identify how a problem will grow. For any practical problem, there will be some bound on the execution time that must be satisfied. And, for the algorithms and multiprocessor families analyzed here, a time constraint tends to be more limiting than the memory constraint. In all cases examined, an indication of how the execution time varies for a memory-constrained model is necessary to interpret the memory-constrained scaled speedup curve. For this reason, we argue that including a bound on the execution time is necessary when defining scaled speedup curves.

For certain problems, the time-constrained model is very pessimistic, indicating that massive parallelism is unlikely to be useful. For other problems, the model indicates that tens of thousands of processors could conceivably be useful, although for that number of processors the analysis is somewhat simplistic. We have ignored costs like loading the program and data and unloading the results. Additionally, there will usually be an upper bound on the size of a problem that is useful to solve, and this will need to be incorporated in a realistic analysis. The examples described here seem to indicate that simple problem and architectural parameters may be sufficient to categorize whether massive parallelism will be useful. For example, the higher the number of space dimensions, the more likely it is that a large number of processors can be used for the hyperbolic examples. For the elliptic examples, the interconnection topology of the multiprocessor determines how many processors to use. Since parabolic problems have traits of both the hyperbolic and elliptic problems, we expect both the number of space dimensions and the interconnection topology to be important when determining how many processors can be used for these problems.

Note that the definition of the scaled speedup curve can be generalized by allowing the algorithm to vary as a function of the number of processors, with only the problem being fixed. Thus, we assume that there is an underlying family of algorithms, each one "best" for a given number of processors. This will not normally be a useful measure for a system administrator. The choice of algorithms is not his to make, and he must evaluate the impact of increasing the number of processors on the current programs. It is also not an interesting generalization for the examples we have analyzed since the standard serial algorithms are easily parallelized. But, the theory described in Worley [23], [24] is algorithm-independent. Thus, for problems to which the theory applies, the qualitative behavior of the generalized time-constrained scaled speedup curve is the same as before.

Acknowledgments. We thank Michael Heath, Charles Romine, and the anonymous referees for their helpful suggestions on the presentation of the material in this paper.

REFERENCES

[1] G. AMDAHL, *The validity of the single processor approach to achieving large scale computing capabilities*, in AFIPS Conference Proceedings, 30 (1967), pp. 483–485.

[2] ————, *Limits of expectation*, International J. Supercomputer Appl., 2 (1988), pp. 88–97.

[3] A. BRANDT, *Guide to multigrid development*, in Multigrid Methods, W. Hackbusch and U. Trottenberg, eds., Springer-Verlag, Berlin, 1982, pp. 220–312.

[4] T. H. DUNIGAN, *Performance of a second generation hypercube*, Tech. Report ORNL/TM-10881, Oak Ridge National Laboratory, Oak Ridge, TN, September 1988.

[5] M. J. FLYNN, *Some computer organizations and their effectiveness*, IEEE Trans. Comput., 21 (1972), pp. 948-960.

[6] G. C. FOX AND S. W. OTTO, *Algorithms for concurrent processors*, Physics Today, 37 (1984), pp. 50–59.

[7] ————, *Concurrent computation and the theory of complex systems*, in Hypercube Multiprocessors 1986, M. T. Heath, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1986, pp. 244–268.

[8] A. GEORGE AND J. W. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[9] 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.

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

[11] M. T. HEATH, ED., *Hypercube Multiprocessors* 1986, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1986.

[12] ————, ED., *Hypercube Multiprocessors* 1987, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

[13] R. W. HOCKNEY AND C. R. JESSHOPE, *Parallel Computers: Architecture, Programming, and Algorithms*, Adam Hilger Ltd., Bristol, UK, 1981.

[14] S. L. JOHNSSON, *Solving tridiagonal systems on ensemble architectures*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 354–392.

[15] L. LAPIDUS AND G. F. PINDER, *Numerical Solution of Partial Differential Equations in Science and Engineering*, John Wiley, New York, 1982.

[16] O. A. MCBRYAN, *Numerical computation on massively parallel hypercubes*, in Hypercube Multiprocessors 1987, M. T. Heath, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987, pp. 706–719.

[17] P. C. MESSINA, *Emerging supercomputer architectures*, in The U.S. Supercomputer Industry, Department of Energy, 1987, Appendix B, pp. B1–B37.

[18] 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, pp. 181–195.

[19] D. A. REED, L. M. ADAMS, AND M. L. PATRICK, *Stencils and problem partitionings: Their influence on the performance of multiple processor systems*, IEEE Trans. Comput., C-36 (1987), pp. 845–858.

[20] R. D. RICHTMEYER AND K. W. MORTON, *Difference Methods for Initial Value Problems*, 2nd ed., John Wiley, New York, 1967.

[21] C. L. SEITZ, *The cosmic cube*, Comm. ACM, 28 (1985), pp. 22–33.

[22] K. STÜBEN AND U. TROTTENBERG, *Multigrid methods: Fundamental algorithms, model problem analysis and applications*, in Multigrid Methods, W. Hackbusch and U. Trottenberg, eds., Springer-Verlag, Berlin, 1982, pp. 1–176.

[23] P. H. WORLEY, *Information requirements and the implications for parallel computation*, Ph.D. thesis, Stanford University, Stanford, CA, June 1988.

[24] ————, *Limits on parallelism in the numerical solution of linear* PDEs, Tech. Report ORNL/TM-10945, Oak Ridge National Laboratory, Oak Ridge, TN, October 1988; SIAM J. Sci. Statist. Comput., submitted.

[25] D. M. YOUNG, *Iterative Solution of Large Linear Systems*, Academic Press, New York, 1971.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 5, pp. 859-872, September 1990

EFFICIENT POLYNOMIAL PRECONDITIONING FOR THE CONJUGATE GRADIENT METHOD*

S. C. EISENSTAT†, J. M. ORTEGA‡, AND C. T. VAUGHAN‡

Abstract. This paper formulates and compares procedures for reducing computation in carrying out m-step or polynomial preconditioning in the conjugate gradient method. These procedures are based on corresponding ones for one-step preconditioning given by Bank and Douglas [*Appl. Numer. Math.*, 1 (1985), pp. 489-492], Conrad and Wallach [*Numer. Math.*, 27 (1979), pp. 371-372], and Eisenstat [*SIAM J. Sci. Statist. Comput.*, 2 (1981), pp. 1-4], and apply, in particular, to SSOR preconditioning. Comparisons are made based on operation counts, storage, and parallel and vector properties, and it is concluded that the Eisenstat procedure is the most effective. Numerical experiments on a parallel computer are also given.

Key words. conjugate gradient method, polynomial preconditioning, Bank-Douglas procedure, Conrad-Wallach procedure, Eisenstat procedure, parallel computers, SSOR

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

1. Introduction.

We consider the system of equations

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

where A is symmetric and positive definite. Conrad and Wallach (1979), Eisenstat (1981), and Bank and Douglas (1985) have proposed ways to reduce the computation in carrying out SSOR and other preconditionings in the conjugate gradient iteration for (1.1). These three approaches were analyzed and compared in Ortega (1988a), but in all of these previous works attention was restricted to "one-step" preconditioners. In order to perform mth degree polynomial preconditioning (Johnson, Miccheli, and Paul (1983)), it is necessary to do m steps of a subsidiary iteration (for example, SSOR) at each conjugate gradient iteration. Adams (1985) used the Conrad-Wallach procedure for SSOR preconditioning in this context.

In the present paper, we will formulate and compare m-step versions of the Bank-Douglas and Eisenstat preconditioning procedures. We also extend the analysis of Ortega (1988a) by comparing storage requirements and parallel properties of the three procedures. Finally, we give the results of numerical experiments performed on an 18 processor Flexible Computer Corp. Flex/32.

2. The m-step preconditioned conjugate gradient procedure.

We begin by reviewing (see, for example, Adams (1985) and Ortega (1988b)) m-step preconditioning. We first write the preconditioned conjugate gradient method for (1.1) in the form ($\mathbf{r}^0=\mathbf{b}-A\mathbf{x}^0$ given, and $k=0,1,\cdots$)

$$M\tilde{\mathbf{r}}^k=\mathbf{r}^k \tag{2.1a}$$

$$\gamma_k=(\tilde{\mathbf{r}}^k,\mathbf{r}^k),\qquad \beta_k=\gamma_k/\gamma_{k-1}\quad(\beta_0=0) \tag{2.1b}$$

$$\mathbf{p}^k=\tilde{\mathbf{r}}^k+\beta_k\mathbf{p}^{k-1}\qquad(\mathbf{p}^{-1}=0) \tag{2.1c}$$

* Received by the editors September 28, 1988; accepted for publication (in revised form) July 12, 1989.

† Department of Computer Science, Yale University, New Haven, Connecticut 06520. The research of this author was supported in part by the Office of Naval Research under contract N00014-86-K-0310 and the National Science Foundation under grant DCR-85-21451.

‡ Department of Applied Mathematics, University of Virginia, Charlottesville, Virginia 22903. The research of this author was supported in part by the National Aeronautics and Space Administration (NASA) under grants NAG-1-242 and NAG-1-46.

$$\alpha_k = \gamma_k/(\mathbf{p}^k, A\mathbf{p}^k) \tag{2.1d}$$

$$\mathbf{r}^{k+1} = \mathbf{r}^k - \alpha_k A\mathbf{p}^k \tag{2.1e}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k \mathbf{p}^k. \tag{2.1f}$$

Here $(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y}$ is the usual inner product and M is the preconditioning matrix. We will be interested in the case in which M has the form

$$M = P(I + H + \cdots + H^{m-1})^{-1} \tag{2.2}$$

where $A = P - Q$ is a splitting of A and $H = P^{-1}Q$. The use of M in (2.1a) is then equivalent to carrying out the subsidiary iteration

$$\tilde{\mathbf{r}}_{i+1}^k = H\tilde{\mathbf{r}}_i^k + P^{-1}\mathbf{r}^k = \tilde{\mathbf{r}}_i^k + P^{-1}(\mathbf{r}^k - A\tilde{\mathbf{r}}_i^k), \qquad i = 0, \cdots, m-1, \qquad \tilde{\mathbf{r}}_0^k = 0 \tag{2.3}$$

for the equation $A\tilde{\mathbf{r}} = \mathbf{r}^k$ and setting $\tilde{\mathbf{r}}^k = \tilde{\mathbf{r}}_m^k$. For polynomial preconditioning (Johnson, Miccheli, and Paul (1983)), $\tilde{\mathbf{r}}^k$ would be obtained by inserting the appropriate multipliers into (2.3). For simplicity, we will focus our attention on m-step preconditioning since the extension to polynomial preconditioning presents no difficulties.

We will consider splittings in which

$$P = (\hat{D} - L)\tilde{D}^{-1}(\hat{D} - L^T) \tag{2.4}$$

where $A = D - L - L^T$ with L strictly lower triangular and D, $\hat{D}$, and $\tilde{D}$ diagonal. In particular, $\hat{D} = \omega^{-1}D$ and $\tilde{D} = 2\hat{D} - D$ gives SSOR preconditioning.

3. The Conrad–Wallach (CW) procedure.

In the CW procedure, the subsidiary iteration (2.3), with P given by (2.4) and $\tilde{D} = 2\hat{D} - D$, is carried out in the "alternating" form

for $i = 0, \cdots, m-1$

$$(\hat{D} - L)\boldsymbol{\delta}_i^k = (\hat{D} - D)\tilde{\mathbf{r}}_i^k + L^T\tilde{\mathbf{r}}_i^k + \mathbf{r}^k, \quad \text{save } L\boldsymbol{\delta}_i^k + \mathbf{r}^k \tag{3.1a}$$

$$(\hat{D} - L^T)\tilde{\mathbf{r}}_{i+1}^k = (\hat{D} - D)\boldsymbol{\delta}_i^k + L\boldsymbol{\delta}_i^k + \mathbf{r}^k, \quad \text{save } L^T\tilde{\mathbf{r}}_{i+1}^k \tag{3.1b}$$

where $\tilde{\mathbf{r}}_0^k = 0$ and $\tilde{\mathbf{r}}^k = \tilde{\mathbf{r}}_m^k$. In this case, the preconditioning step, (2.1a), is replaced by (3.1). The "saves" in (3.1), which are the key to the Conrad–Wallach procedure, are based on the observation that in (3.1a), $L\boldsymbol{\delta}_i^k + \mathbf{r}^k$ is computed in the course of the solution for $\boldsymbol{\delta}_i^k$ and can be saved for use in (3.1b). Similarly, $L^T\tilde{\mathbf{r}}_{i+1}^k$ can be saved in (3.1b) for use in (3.1a) at the next iteration.

The evaluation of $A\mathbf{p}^k$ for (2.1) is achieved by

$$A\mathbf{p}^k = D\mathbf{p}^k - L\mathbf{p}^k - L^T\mathbf{p}^k. \tag{3.2}$$

From (2.1c)

$$L^T\mathbf{p}^k = L^T\tilde{\mathbf{r}}^k + \beta_k L^T\mathbf{p}^{k-1} \tag{3.3}$$

and $L^T\tilde{\mathbf{r}}^k$ is the quantity saved in (3.1b) the last time through the loop. If we assume that $L^T\mathbf{p}^{k-1}$ has been saved from the last conjugate gradient iteration, then the evaluation of $L^T\mathbf{p}^k$ by (3.3) saves the multiplication by L^T in (3.2). This is a more efficient form of the Conrad–Wallach procedure than the straightforward one analyzed in Ortega (1988a) for $m = 1$. Another savings results for $i = 0$ by setting $L\boldsymbol{\delta}_0^k + \mathbf{r}^k = \hat{D}\boldsymbol{\delta}_0^k$, which is true by (3.1a), in (3.1b) to obtain

$$(\hat{D} - L^T)\tilde{\mathbf{r}}_1^k = \tilde{D}\boldsymbol{\delta}_0^k, \quad \text{save } L^T\tilde{\mathbf{r}}_1^k. \tag{3.4}$$

This would be used in place of (3.1b) for $i = 0$ and saves one vector addition over forming the right-hand side of (3.1b).

4. The Eisenstat (E) procedure. The E procedure is based on the change of variable

$$\hat{\mathbf{x}} = (\hat{D} - L^T)\mathbf{x}, \qquad \hat{\mathbf{b}} = (\hat{D} - L)^{-1}\mathbf{b}. \tag{4.1}$$

The conjugate gradient iteration is then applied to the system

$$\hat{A}\hat{\mathbf{x}} = \hat{\mathbf{b}}, \qquad \hat{A} = (\hat{D} - L)^{-1}A(\hat{D} - L^T)^{-1} \tag{4.2}$$

and for one-step preconditioning can be written in the form ($\hat{\mathbf{r}}^0 = (\hat{D} - L)^{-1}\mathbf{r}^0$, $k = 0, 1, \cdots$)

$$\mathbf{t}^k = \tilde{D}\hat{\mathbf{r}}^k \tag{4.3a}$$

$$\gamma_k = (\mathbf{t}^k, \hat{\mathbf{r}}^k), \qquad \beta_k = \gamma_k/\gamma_{k-1} \quad (\beta_0 = 0) \tag{4.3b}$$

$$\hat{\mathbf{p}}^k = \mathbf{t}^k + \beta_k\hat{\mathbf{p}}^{k-1} \qquad (\hat{\mathbf{p}}^{-1} = 0) \tag{4.3c}$$

$$(\hat{D} - L^T)\mathbf{p}^k = \hat{\mathbf{p}}^k \tag{4.3d}$$

$$\mathbf{z}^k = \hat{\mathbf{p}}^k - \bar{D}\mathbf{p}^k \qquad (\bar{D} = 2\hat{D} - D) \tag{4.3e}$$

$$(\hat{D} - L)\mathbf{w}^k = \mathbf{z}^k \tag{4.3f}$$

$$\mathbf{a}^k = \mathbf{p}^k + \mathbf{w}^k \; (= \hat{A}\hat{\mathbf{p}}^k) \tag{4.3g}$$

$$\alpha_k = \gamma_k/(\hat{\mathbf{p}}^k, \mathbf{a}^k) \tag{4.3h}$$

$$\hat{\mathbf{r}}^{k+1} = \hat{\mathbf{r}}^k - \alpha_k\mathbf{a}^k \tag{4.3i}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k\mathbf{p}^k. \tag{4.3j}$$

The key to the E procedure is the evaluation of $\hat{A}\hat{\mathbf{p}}$. Since, with $\bar{D} = 2\hat{D} - D$,

$$A = (\hat{D} - L) + (\hat{D} - L^T) - \bar{D} \tag{4.4}$$

we have

$$\hat{A}\hat{\mathbf{p}} = (\hat{D} - L^T)^{-1}\hat{\mathbf{p}} + (\hat{D} - L)^{-1}[\hat{\mathbf{p}} - \bar{D}(\hat{D} - L^T)^{-1}\hat{\mathbf{p}}]. \tag{4.5}$$

Thus, with $\mathbf{p} = (\hat{D} - L^T)^{-1}\hat{\mathbf{p}}$, we need to solve the systems

$$(\hat{D} - L^T)\mathbf{p} = \hat{\mathbf{p}}, \qquad (\hat{D} - L)\mathbf{w} = \hat{\mathbf{p}} - \bar{D}\mathbf{p}. \tag{4.6}$$

This is done in (4.3d) and (4.3f), and then $\mathbf{a}^k = \hat{A}\hat{\mathbf{p}}^k$ is computed in (4.3g). The preconditioning has been reduced to a multiplication by $\tilde{D}$, which is (4.3a), and the other steps in (4.3) correspond to those of (2.1).

We now wish to incorporate m-step preconditioning into (4.3). Thus, we take m steps of the subsidiary iteration (2.3) with P given by (2.4). In the context of the E procedure, it is natural to transform the subsidiary iteration by the same change of variable, (4.1), used to obtain (4.3). Hence, with

$$\mathbf{t}_i^k = (\hat{D} - L^T)\tilde{\mathbf{r}}_i^k, \qquad \hat{\mathbf{r}}^k = (\hat{D} - L)^{-1}\mathbf{r}^k \tag{4.7}$$

(2.3) becomes

$$\mathbf{t}_{i+1}^k = \mathbf{t}_i^k + \tilde{D}(\hat{\mathbf{r}}^k - \hat{A}\mathbf{t}_i^k), \qquad i = 0, \cdots, m-1, \qquad \mathbf{t}_0^k = 0 \tag{4.8}$$

where $\hat{A}$ is given by (4.2).

In order to carry out the subsidiary iteration (4.8) efficiently, we need to avoid the multiplication by $\hat{A}$ and this can be done in exactly the same way as in the E procedure (3.3); indeed, we just apply the formulas (4.3d)-(4.3g) with $\mathbf{t}_i^k$ replacing $\hat{\mathbf{p}}^k$. (In light of (4.7), we also replace $\mathbf{p}^k$ by $\tilde{\mathbf{r}}_i^k$.) Then the subsidiary iteration takes the form

$$\mathbf{t}_1^k = \tilde{D}\hat{\mathbf{r}}^k. \tag{4.9a}$$

For $i = 1$ to $m-1$

$$(\hat{D} - L^T)\tilde{\mathbf{r}}_i^k = \mathbf{t}_i^k \tag{4.9b}$$

$$\mathbf{z}_i^k = \mathbf{t}_i^k - \bar{D}\tilde{\mathbf{r}}_i^k \tag{4.9c}$$

$$(\hat{D} - L)\mathbf{w}_i^k = \mathbf{z}_i^k \tag{4.9d}$$

$$\mathbf{a}_i^k = \tilde{\mathbf{r}}_i^k + \mathbf{w}_i^k \ (= \hat{A}\mathbf{t}_i^k) \tag{4.9e}$$

$$\mathbf{t}_{i+1}^k = \mathbf{t}_i^k + \tilde{D}(\hat{\mathbf{r}}^k - \mathbf{a}_i^k). \tag{4.9f}$$

Finally, at the termination of the i loop, we set

$$\mathbf{t}^k = \mathbf{t}_m^k. \tag{4.9g}$$

Thus, for m-step preconditioning, we replace (4.3a) by (4.9). In the case that $m = 1$, (4.9) reduces to the original (4.3a).

We note that in the case that $\bar{D} = \tilde{D} = 2\hat{D} - D$ we can replace (4.9e) and (4.9f) by

$$\mathbf{t}_{i+1}^k = \mathbf{z}_i^k + \tilde{D}(\hat{\mathbf{r}}^k - \mathbf{w}_i^k) \tag{4.9h}$$

which results from substituting $\mathbf{a}_i^k$ into (4.9f) and then using (4.9c).

5. The Bank–Douglas (BD) procedure.

Bank and Douglas (1985) gave another approach to the efficient implementation of one-step preconditioning in the conjugate gradient method. Ortega (1988a) gave the BD procedure in the following slightly modified form (Bank and Douglas assumed that $\tilde{D} = \hat{D}$). Here, $k = 0, 1, \cdots$, and $\mathbf{s}^0 = \mathbf{b} - (D - L^T)\mathbf{x}^0$.

$$(\hat{D} - L)\hat{\mathbf{r}}^k = \mathbf{s}^k + L\mathbf{x}^k \tag{5.1a}$$

$$\mathbf{t}^k = \tilde{D}\hat{\mathbf{r}}^k \tag{5.1b}$$

$$(\hat{D} - L^T)\tilde{\mathbf{r}}^k = \mathbf{t}^k \tag{5.1c}$$

$$\gamma_k = (\mathbf{t}^k, \hat{\mathbf{r}}^k), \qquad \beta_k = \gamma_k / \gamma_{k-1} \quad (\beta_0 = 0) \tag{5.1d}$$

$$\mathbf{p}^k = \tilde{\mathbf{r}}^k + \beta_k \mathbf{p}^{k-1} \qquad (\mathbf{p}^{-1} = 0) \tag{5.1e}$$

$$\mathbf{v}^k = \mathbf{t}^k + \beta_k \mathbf{v}^{k-1} - (\hat{D} - D)\tilde{\mathbf{r}}^k \qquad (\mathbf{v}^{-1} = 0) \tag{5.1f}$$

$$\alpha_k = \tfrac{1}{2}\gamma_k / (\mathbf{p}^k, \mathbf{v}^k - \tfrac{1}{2}D\mathbf{p}^k) \tag{5.1g}$$

$$\mathbf{s}^{k+1} = \mathbf{s}^k - \alpha_k \mathbf{v}^k \tag{5.1h}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k \mathbf{p}^k. \tag{5.1i}$$

In the original BD procedure, $\tilde{D} = \hat{D}$. In this case (5.1a) is changed to

$$(\hat{D} - L)\hat{\mathbf{r}}^k = \mathbf{s}^k + L\mathbf{x}^k, \quad \text{save } \mathbf{t}^k = \mathbf{s}^k + L(\mathbf{x}^k + \hat{\mathbf{r}}^k), \tag{5.2}$$

which eliminates the need for (5.1b). In either (5.1a) or (5.2), a key observation is that $L\mathbf{x}^k$ is not evaluated explicitly; rather, $L(\mathbf{x}^k + \hat{\mathbf{r}}^k)$ is obtained during the course of the solution.

The steps (5.1a)-(5.1c) carry out one-step preconditioning and we wish to replace these by m-step preconditioning. A natural approach to this is to use a way proposed by Bank and Douglas to carry out stationary iterations (see also Ortega (1988a)). This gives the following version of the subsidiary iteration (2.3), which would be inserted after (5.1c) and used together with (5.1a)-(5.1c) when $m>1$.

$$\mathbf{s}_0^k=\mathbf{r}^k, \qquad \tilde{\mathbf{r}}_1^k=\tilde{\mathbf{r}}^k, \qquad \mathbf{t}_0^k=\mathbf{t}^k, \qquad \boldsymbol{\delta}_1^k=\tilde{\mathbf{r}}^k$$

For $i=1$ to $m-1$

(5.3a) $$\mathbf{s}_i^k=\mathbf{s}_{i-1}^k-\mathbf{t}_{i-1}^k+(\hat{D}-D)\boldsymbol{\delta}_i^k$$

(5.3b) $$(\hat{D}-L)\boldsymbol{\delta}_{i+1/2}^k=\mathbf{s}_i^k+L\tilde{\mathbf{r}}_i^k$$

(5.3c) $$\mathbf{t}_i^k=\tilde{D}\boldsymbol{\delta}_{i+1/2}^k$$

(5.3d) $$(\hat{D}-L^T)\boldsymbol{\delta}_{i+1}^k=\mathbf{t}_i^k$$

(5.3e) $$\tilde{\mathbf{r}}_{i+1}^k=\tilde{\mathbf{r}}_i^k+\boldsymbol{\delta}_{i+1}^k.$$

At the end of (5.3) we would set $\tilde{\mathbf{r}}^k=\tilde{\mathbf{r}}_m^k$ and $\mathbf{t}^k=\mathbf{t}_{m-1}^k$. Again, if $\tilde{D}=\hat{D}$, (5.3c) would be eliminated and (5.3b) modified as in (5.2).

Unfortunately, this version of the subsidiary iteration has several problems. First, the initial condition and (5.3a) call for $\mathbf{s}_0^k=\mathbf{r}^k$, which is not computed in (5.1). Thus, it is necessary to compute $\mathbf{r}^k=\mathbf{s}^k+L\mathbf{x}^k$ at the expense of an additional multiplication by L and a vector addition. Once $\mathbf{r}^k$ is computed, we can overcome a second problem, namely, that the computation of γ_k by (5.1d) is not correct if $m>1$. Hence, we revert to the original $\gamma_k=(\tilde{\mathbf{r}}^k,\mathbf{r}^k)$ of (2.1b). A third problem is the following. In the one-step procedure (5.1), a key relation is that

(5.4) $$\mathbf{v}^k=A\mathbf{p}^k+L\mathbf{p}^k\ (=(D-L^T)\mathbf{p}^k)$$

so that $\mathbf{v}^k$ is a "partial" $A\mathbf{p}^k$. It is shown in Ortega (1988a) that the $\mathbf{v}^k$ defined by (5.1f) satisfy (5.4), and the key to the equality is the relation (5.1c) between $\tilde{\mathbf{r}}^k$ and $\mathbf{t}^k$. However, if $\tilde{\mathbf{r}}^k=\mathbf{r}_m^k$ is produced by (5.3), this relation no longer holds; instead, we have (5.3d) and (5.3e). We can partially circumvent this problem by replacing (5.1f) by

(5.5) $$\mathbf{v}^k=(\hat{D}-L^T)\tilde{\mathbf{r}}^k+\beta_k\mathbf{v}^{k-1}-(\hat{D}-D)\tilde{\mathbf{r}}^k.$$

The relation (5.4) can then be shown to hold but the quantity $(\hat{D}-L^T)\tilde{\mathbf{r}}^k$ is not produced in (5.3). We can obtain it by combining (5.3d) and (5.3e) to give

(5.6) $$\boldsymbol{\sigma}_{i+1}^k=\mathbf{t}_i^k+\boldsymbol{\sigma}_i^k, \qquad i=1,2,\cdots,m-1$$

where $\boldsymbol{\sigma}_i^k=(\hat{D}-L^T)\tilde{\mathbf{r}}_i^k$ and $\boldsymbol{\sigma}_1^k=\mathbf{t}_0^k$. Thus, we can compute the $\boldsymbol{\sigma}_i^k$ at the cost of one more vector addition per inner iteration.

As an alternative to using the stationary Bank-Douglas iteration (5.3) in conjunction with (5.1), we can implement the preconditioning step (2.1a) by (5.3). In this case, the loop for (5.3) would run from $i=0$ to $m-1$ and the initial conditions would be

$$\mathbf{s}_{-1}^k=\mathbf{r}^k, \qquad \tilde{\mathbf{r}}_0^k=\mathbf{t}_{-1}^k=\boldsymbol{\delta}_0^k=0$$

which imply, by (5.3a), that $\mathbf{s}_0^k=\mathbf{r}^k$. Note that $\mathbf{r}^k$ is computed in (2.1) so no extra work for it is needed. We will call this procedure BDS1_m. A variation of this, which we call BDS2_m, allows us to save a multiplication by L^T in the evaluation of $A\mathbf{p}^k$, as in the CW procedure. In this case, we would use (5.6) to compute $\boldsymbol{\sigma}_m=(\hat{D}-L^T)\tilde{\mathbf{r}}^k$. Then

(5.7) $$(\hat{D}-L^T)\mathbf{p}^k=(\hat{D}-L^T)\tilde{\mathbf{r}}^k+\beta_k(\hat{D}-L^T)\mathbf{p}^{k-1}$$

and

$$Ap^k = (\hat{D} - L^T)\mathbf{p}^k - (\hat{D} - D)\mathbf{p}^k - L\mathbf{p}^k. \tag{5.8}$$

Here, there is a trade-off between saving the multiplication by L^T and the extra operations from (5.6) and (5.7), as well as the storage for $(\hat{D} - L^T)\mathbf{p}^{k-1}$ between iterations.

Still another alternative is to use the stationary Eisenstat procedure (4.9) in (5.1); that is, we replace (5.1b) in the Bank-Douglas procedure (5.1) by (4.9). Now the relation $\mathbf{t}^k = (\hat{D} - L^T)\tilde{\mathbf{r}}^k$ is automatically satisfied, by (5.1c), so that (5.4) holds. We will call this combination the BDE$_m$ procedure.

6. Comparison of the procedures. We first compare the operation counts. It was shown in Ortega (1988a) that the operation counts of the one-step BD and E procedures are

$$\text{BD} = 2l + 5d + 5v + 4l_t + 2i_p \tag{6.1}$$

$$\text{E} = 2l + 4d + 4v + 3l_t + 2i_p \tag{6.2}$$

where l is a multiplication by either L or L^T, d is a multiplication by a diagonal matrix, v is a vector addition, i_p is an inner product, and l_t is a linked triad (vector + scalar times vector). Scalar operations such as divisions have been ignored in these counts.

For the one-step Conrad-Wallach procedure, using (3.3) in the evaluation of $A\mathbf{p}^k$ and (3.4) in place of (3.1b), the operation count is

$$\text{CW} = 3l + 4d + 4v + 4l_t + 2i_p \tag{6.3}$$

which has one less l, one less v, and one more l_t than the count without using (3.3) and (3.4) given in Ortega (1988a). It was also shown in Ortega (1988a) that the operation count of the stationary CW procedure (3.1) is $2l + 4d + 5v$ per iteration, except for the first iteration in which $\tilde{\mathbf{r}}_0^k = 0$. However, this first iteration is included in (6.3) for the one-step CW procedure so that the m-step CW procedure has an operation count of

$$\begin{aligned} \text{CW}_m &= 3l + 4d + 4v + 4l_t + 2i_p + (m-1)(2l + 4d + 5v) \\ &= (2m+1)l + 4md + (5m-1)v + 4l_t + 2i_p. \end{aligned} \tag{6.4}$$

For the E procedure, the operation count for (4.9b)-(4.9f) is

$$(d + l + v) + (d + v) + (d + l + v) + (3v + d) = 2l + 4d + 6v \tag{6.5}$$

per iteration. Since, again, the first step of the preconditioning is included in (6.2), the operation count for the m-step E procedure is

$$\begin{aligned} E_m &= 2l + 4d + 4v + 3l_t + 2i_p + (m-1)(2l + 4d + 6v) \\ &= 2ml + 4md + (6m-2)v + 3l_t + 2i_p. \end{aligned} \tag{6.6}$$

For the BDE procedure, the operation count for each stationary iteration is again given by (6.5) so for the m-step procedure we have

$$\text{BDE}_m = 2ml + (4m+1)d + (6m-1)v + 4l_t + 2i_p. \tag{6.7}$$

The operation count for the subsidiary iteration (5.3) and including (5.6) is $2l + 4d + 7v$. We also need to add one l and one v to (6.1) to account for the computation of $\mathbf{r}^k = \mathbf{s}^k + L\mathbf{x}^k$, as discussed previously. Thus, the operation count for the m-step BD procedure is

$$\begin{aligned} \mathrm{BD}_m &= 3l + 5d + 6v + 4l_t + 2i_p + (m-1)(2l + 4d + 7v) \\ &= (2m+1)l + (4m+1)d + (7m-1)v + 4l_t + 2i_p. \end{aligned} \tag{6.8}$$

Note that (6.8) does not reduce to (6.1) if $m = 1$, because of the computation of $\mathbf{r}^k$.

For the $\mathrm{BDS1}_m$ procedure, the operation count for each iteration of (5.3) after the first is $2l + 4d + 6v$, since (5.6) is now not used. For the first iteration of (5.3) (that is, $i = 0$) the initial conditions imply that two v and one d may be saved in (5.3a), one v in (5.3b), and one v in (5.3e). Combining these counts with the remaining steps of (2.1) then gives

$$\begin{aligned} \mathrm{BDS1}_m &= (2l + d + 2v + 3l_t + 2i_p) + (2l + 3d + 2v) + (m-1)(2l + 4d + 6v) \\ &= (2m+2)l + 4md + (6m-2)v + 3l_t + 2i_p. \end{aligned} \tag{6.9}$$

For the $\mathrm{BDS2}_m$ procedure, which uses (5.6)-(5.8), each iteration of (5.3) and (5.6) after the first requires $2l + 4d + 7v$, whereas the first requires $2l + 3d + 2v$ since (5.6) is not needed for $i = 0$. One l in the conjugate gradient iteration is replaced by an additional l_t for (5.7). Hence

$$\mathrm{BDS2}_m = (2m+1)l + 4md + (7m-3)v + 4l_t + 2i_p. \tag{6.10}$$

For comparison we also give the operation count of a straightforward (SF) implementation using (2.1) with (3.1) but not with (3.4). If (3.4) is used, the count can be lowered by one l and two v. We summarize the operation counts in Table 1. The m-step E procedure is slightly more efficient than the BDE procedure, and both are more efficient than the others.

TABLE 1
Operation counts for m-step preconditioning.

SF	$(4m+1)l + 4md + 6mv + 3l_t + 2i_p$
CW	$(2m+1)l + 4md + (5m-1)v + 4l_t + 2i_p$
E	$2ml + 4md + (6m-2)v + 3l_t + 2i_p$
BDE	$2ml + (4m+1)d + (6m-1)v + 4l_t + 2i_p$
BD	$(2m+1)l + (4m+1)d + (7m-1)v + 4l_t + 2i_p$
BDS1	$(2m+2)l + 4md + (6m-2)v + 3l_t + 2i_p$
BDS2	$(2m+1)l + 4md + (7m-3)v + 4l_t + 2i_p$

We next compare the amount of storage that each method requires. First, consider the conjugate gradient method with no preconditioning; this is (2.1) with (2.1a) deleted and $\tilde{\mathbf{r}} = \mathbf{r}$. Clearly, storage is required for $\mathbf{p}$, $\mathbf{r}$, and $\mathbf{x}$, which are saved from iteration to iteration. Temporary storage is also required for $A\mathbf{p}$. (On some vector machines, additional temporary storage may be required in the formation of $A\mathbf{p}$, depending on the storage of A.) We assume that $\mathbf{p}^k$ overwrites $\mathbf{p}^{k-1}$ in (2.1c) and similarly for $\mathbf{r}^{k+1}$ and $\mathbf{x}^{k+1}$. Hence, ignoring the few scalar positions required, the total storage is, in addition to the matrix A and the right-hand side $\mathbf{b}$,

$$S_{CG} = 4 \ n\text{-vectors}. \tag{6.11}$$

The storage required for A is dependent on whether symmetry is utilized; this may affect the efficiency of the codes, especially on parallel and vector machines. Also, in

(6.11) and in all the other algorithms, one of the temporary vectors can occupy the location of $\mathbf{b}$ if so desired. Since all of the algorithms require storage for A and $\mathbf{b}$, we will give in the following the additional storage needed, as in (6.11).

For the CW procedure, we also need three vectors of storage for $\hat{D}^{-1}$, $\tilde{D}$, and $\hat{D}-D$. Again, $\mathbf{p}$, $\mathbf{r}$, and $\mathbf{x}$ are saved from iteration to iteration, as is $L^T\mathbf{p}$ to use (3.3). Two other temporary vectors, $\mathbf{s}_1$ and $\mathbf{s}_2$, are also needed for (3.1) and (3.4). Once the preconditioning is complete, $\tilde{\mathbf{r}}$ occupies $\mathbf{s}_1$, and $L^T\tilde{\mathbf{r}}$ occupies $\mathbf{s}_2$. Then $L^T\mathbf{p}^k$ replaces $L^T\mathbf{p}^{k-1}$ according to (3.3). At this point, $\mathbf{s}_1$ and $\mathbf{s}_2$ are free and the remainder of (2.1) is equivalent to the conjugate gradient iteration in storage, with the formation of $A\mathbf{p}$ by (3.2) using $\mathbf{s}_2$. For m-step preconditioning, consider a typical step of (3.1). $\tilde{\mathbf{r}}$ and $L^T\tilde{\mathbf{r}}$ have been saved in $\mathbf{s}_1$ and $\mathbf{s}_2$ from the previous step. Then $(\hat{D}-D)\tilde{\mathbf{r}}$ overwrites $\mathbf{s}_1$ and the sum of this and $L^T\tilde{\mathbf{r}}$ overwrites $\mathbf{s}_2$. Next, $L\boldsymbol{\delta}+\mathbf{r}$ overwrites $\mathbf{s}_1$ and $\boldsymbol{\delta}$ overwrites $\mathbf{s}_2$. $(\hat{D}-D)\boldsymbol{\delta}$ can then overwrite $\mathbf{s}_2$ and the sum with $L\boldsymbol{\delta}+\mathbf{r}$ can overwrite $\mathbf{s}_1$. Finally, $L^T\tilde{\mathbf{r}}$ can overwrite $\mathbf{s}_2$ and $\tilde{\mathbf{r}}$ can overwrite $\mathbf{s}_1$. Thus, the storage for the CW procedure is, assuming that D is also stored,

$$S_{\mathrm{CW}} = 9\ n\text{-vectors}. \tag{6.12}$$

Along the same lines, Vaughan (1988) has given a detailed analysis of the storage requirements of the other procedures. These are summarized in Table 2, which shows that the E procedure has the minimum storage requirements.

TABLE 2
Storage vectors for m-step preconditioning.

Method	Storage	Comments
CW	9	Provided that D is stored
E	7	Provided that D is stored
BDE	11	
BD	14	10 for one-step preconditioning
BDS1	10	
BDS2	12	

Finally, we consider the properties of the procedures on parallel and vector machines. All have vector and parallel properties as good as the conjugate gradient iteration itself, with the possible exception of the solution of the systems with coefficient matrices $\hat{D}-L$ and $\hat{D}-L^T$. The major cost of the solution of the triangular systems, as well as multiplication by A, has been incorporated into the operation counts by the cost, l, of multiplication by a triangular matrix. But on vector and parallel machines there can be an important difference between multiplication by a triangular matrix and solution of a triangular system. Thus, on these machines, the additional l's in the operation counts, beyond the $2ml$ needed for solution of the systems, may not be as significant since they reflect matrix multiplication. This may have the effect of making the different procedures perform more uniformly than their operation counts would indicate.

7. Multicolor orderings and SSOR. One approach to solving the triangular systems on vector and parallel machines is by means of multicolor orderings (see, e.g., Ortega (1988b)), and the use of such orderings changes the operation counts of some of the procedures, as we next discuss. With a multicolor ordering with c colors, the matrix

A has the block form

$$A=\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1c} \\ \vdots & & & \vdots \\ A_{c1} & \cdots & & A_{cc} \end{bmatrix} \tag{7.1}$$

where, if the number of colors is chosen correctly, the diagonal blocks A_{ii} will be diagonal or have some simple block diagonal form. This allows SSOR to be carried out in a vector or parallel form since the diagonal blocks are easily inverted and the main effort in the SSOR iterations is multiplication by the off-diagonal blocks.

Whatever the structure of the diagonal blocks in (7.1), there is an economy of the following form associated with block SSOR. Consider first block Gauss-Seidel ($\omega = 1$), and let $\mathbf{x}^k$ be the current block Gauss-Seidel iterate and $\boldsymbol{\delta}^k$ be the result of the forward block Gauss-Seidel sweep. Then

$$\boldsymbol{\delta}_i^k = A_{ii}^{-1}\left[\mathbf{b}_i - \sum_{j=1}^{i-1} A_{ij}\boldsymbol{\delta}_i^k - \sum_{j=i+1}^{c} A_{ij}\mathbf{x}_j^k\right], \qquad i=1,\cdots,c$$

$$\mathbf{x}_i^{k+1} = A_{ii}^{-1}\left[\mathbf{b}_i - \sum_{j=i+1}^{c} A_{ij}\mathbf{x}_j^{k+1} - \sum_{j=1}^{i-1} A_{ij}\boldsymbol{\delta}_j^k\right], \qquad i=c, c-1,\cdots,1.$$

In particular,

$$\mathbf{x}_c^{k+1} = A_{cc}^{-1}\left[\mathbf{b}_c - \sum_{j=1}^{c-1} A_{cj}\boldsymbol{\delta}_j^k\right] = \boldsymbol{\delta}_c^k \tag{7.2}$$

so that the computation of $\mathbf{x}_c^{k+1}$ may be bypassed, saving the multiplications by $A_{c1},\cdots,A_{c,c-1}$. Similarly at the beginning of the next iteration

$$\boldsymbol{\delta}_1^{k+1} = \mathbf{x}_1^{k+1} \tag{7.3}$$

and the computation of $\boldsymbol{\delta}_1^{k+1}$ may be bypassed, saving the multiplications by $A_{12},\cdots,A_{1c}$. In the case of a multicolor ordering in which the blocks A_{ii} are diagonal, if the nonzero off-diagonal elements of A are spread uniformly over the A_{ij} (as is approximately the case for our test problems of the next section), then the savings in (7.2) amounts to

$$\frac{c-1}{c(c-1)/2}\,l = \frac{2}{c}\,l \tag{7.4}$$

where, again, l is the work of a multiplication by L or L^T. Thus, if $c=2$ (red/black ordering), the savings is l while for $c=3$ and $c=4$, as used in the next section, the savings are $\frac{2}{3}l$ and $\frac{1}{2}l$, respectively. On subsequent steps of the iteration, these savings are doubled by use of (7.3).

If $\omega \neq 1$, the same savings in l hold. It is no longer the case that $\mathbf{x}_c^{k+1} = \boldsymbol{\delta}_c^k$ but by using $\hat{\omega} = \omega(2-\omega)$ in the step to compute $\boldsymbol{\delta}_c^k$, we obtain $\mathbf{x}_c^{k+1}$ instead, as shown, for example, in Ortega (1988b, p. 175). (Note that we do not obtain $\boldsymbol{\delta}_c^k$, which is not needed. Note, also, that we require a separate copy of $\hat{D}$, corresponding to $\hat{\omega}$, to be stored.) The analogous situation holds in obtaining $\mathbf{x}_1^{k+2}$ on the next SSOR step. Thus, in the first SSOR step we save $1/2c$ of the computations leading to the contribution $2d+2v$ in the SF operation count and $1/c$ of $4d+6v$ in each subsequent SSOR step. Combining these savings with those in l, the operation count given in Table 1 for the straightforward procedure is changed for multicolor orderings to

$$\left(5-\frac{2}{c}\right)l+\left(4-\frac{4}{c}\right)(m-1)l+\left(4-\frac{4}{c}\right)md+\frac{5}{2c}\,d+\left(6-\frac{6}{c}\right)mv+\frac{4}{c}\,v+3l_t+2i_p. \tag{7.5}$$

The CW procedure already saves the multiplications by *all* of the A_{ij}, $i>j$, on the backward step, not just the A_{ci} as indicated in (7.2), and obtains only the savings in the vector operations necessary to obtain $\mathbf{x}_c^{k+1}$ instead of $\boldsymbol{\delta}_c^k$, as discussed in the previous paragraph. In the other procedures, there are no savings. Thus, the use of multicolor orderings reduces primarily the operation count of the straightforward procedure and, slightly, that of the CW procedure.

8. Numerical experiments. In this section, we report on numerical experiments for SSOR preconditioning on an 18 processor Flexible Computer Corp. Flex/32, a local memory machine that uses a shared memory for communication. Results are given for two model problems. The first is $u_{xx}+u_{yy}+u_{xy}=0$, which is Laplace's equation with a mixed derivative term added. The domain is the unit square with Dirichlet boundary conditions, and the equation is discretized using standard second order finite differences. (See Ortega (1988b) for a detailed description.) The second problem is a two-dimensional plane stress problem in which a square plate (with Young's modulus 3.5904 and Poisson ratio 0.32) is fastened to a rigid body along one side, and a unit point load is applied to the top of the other side. It is discretized with triangular finite elements using linear basis functions and has two unknowns at each grid point. (See Adams (1982) for a detailed description.) In both problems, the initial approximation to the solution is zero, and the convergence test is $(\mathbf{r}^k, \tilde{\mathbf{r}}^k) \leqq 10^{-6}$. A relaxation parameter of $\omega=1$ is used in all the experiments. If $\omega=1$, some savings are available (for example, $\hat{D}-D=0$ in the CW procedure (3.1)) but they have not been used and the codes are all written for general ω.

For implementation of the methods on p processors, the grid of unknowns is divided into p rectangles, each of which is assigned to a processor. The linked triads and the vector adds can then be done in parallel with each processor calculating the part that corresponds to the unknowns that it has been assigned. The partial inner products done by each processor are combined in global memory. The matrix-vector multiply requires synchronization of the processors after the calculation of $\mathbf{p}$, which is stored in the shared memory, to ensure using the current value of $\mathbf{p}$ for the matrix-vector multiply.

In order to implement SSOR preconditioning in parallel, a multicolor ordering of the unknowns has been used, as discussed in the previous section. The SSOR passes require synchronization after each color of unknown is updated so that these updated values can be used to update the unknowns of the remaining colors. The mixed derivative problem uses a four-color ordering. Because there are two unknowns at each grid point, the plane stress problem requires six colors in order that the diagonal blocks of (7.1) be diagonal. We have chosen to use a three-color ordering in which both of the unknowns at a grid point are the same color. In this case, the matrix D is block diagonal with 2×2 blocks and we have reformulated (3.1a) as

$$(I-\hat{D}^{-1}L)\boldsymbol{\delta}_i^k=(I-\hat{D}^{-1}D)\tilde{\mathbf{r}}_i+\hat{D}^{-1}(L^T\tilde{\mathbf{r}}_i^k+\mathbf{r}^k)$$

and similarly for (3.1b). For SSOR preconditioning $I-\hat{D}^{-1}D=(1-\omega)I$, which allows a savings in computation. The matrices $\hat{D}^{-1}L$ and $\hat{D}^{-1}L^T$ are not formed explicitly, only $\hat{D}^{-1}$. The analogous savings are obtained in the E procedure (and, hence, the BDE procedure) by reformulating (4.9) in terms of $\boldsymbol{\tau}_i^k=\hat{D}^{-1}\mathbf{t}_i^k$.

We first give in Table 3 times for serial codes on a single processor, using the natural ordering of the unknowns. The final three columns are the ratio of times to that of a straightforward implementation of one-step SSOR preconditioning that reflects the operation count in Table 1. The number of unknowns corresponds to 60×60, 86×86, and 120×120 grids for the mixed derivative problem, and 32×32, 44×44, and

TABLE 3
Serial one-step preconditioning. Natural ordering.

			Time relative to SF		
Problem	Unknowns	Iterations	CW	BD	E
Mixed derivative	3600	44	0.894	0.790	0.706
	7396	62	0.893	0.790	0.706
	14400	85	0.892	0.787	0.701
Plane stress	2048	59	0.739	0.576	0.564
	3872	80	0.727	0.566	0.555
	8192	115	0.726	0.564	0.552

64×64 grids for the plane stress problem. (Recall that there are two unknowns per grid point for this problem.) As Table 3 shows, the E procedure was the fastest, although the BD procedure was a close second on the plane stress problem. Note that all procedures performed relatively much better on the plane stress problem. This is because the coefficient matrix of the plane stress problem is much less sparse; consequently, the l's in the operation counts weigh relatively more and it is primarily l's that are saved by all of the procedures, relative to the straightforward (SF) implementation.

For m-step preconditioning, we have implemented the CW, E, BD, and BDE procedures, but not BDS1 and BDS2 since their behavior should be similar to the BD procedure. In the implementation of the E and BDE procedures, we have used (4.10) in place of (4.9e) and (4.9f) since $\bar{D}=\tilde{D}$ for SSOR preconditioning. Serial results for $m=2$, 4, and 6 are shown in Table 4. The ratios shown in Table 3 are relatively insensitive to problem size and the same is true of m-step preconditioning; consequently, we have given results only for the smallest problems in Table 4. Again, the E procedure is the best, as predicted, and the BDE procedure is a close second.

As in Table 3, the results in Table 4 are given relative to the SF implementation. The number of iterations decreases as m increases but not rapidly enough to compensate for the additional time required for the preconditioning. For example, the time for the plane stress problem with 2048 unknowns using the CW procedure was 322s for $m=4$ compared with 216s for $m=1$. However, addition of the polynomial parameters reduced the time for $m=4$ to 213s. In practice, of course, we would use polynomial preconditioning instead of just m-step, but that adds nothing to this study of the comparison of different ways to save computation.

TABLE 4
Serial m-step preconditioning. Natural ordering.

			Time relative to SF			
Problem	m	Iterations	CW	BD	BDE	E
	2	32	0.834	0.951	0.773	0.745
Mixed derivative	4	23	0.797	0.929	0.768	0.737
3600	6	19	0.778	0.916	0.765	0.732
	2	43	0.671	0.723	0.577	0.572
Plane stress	4	31	0.630	0.688	0.580	0.574
2048	6	25	0.617	0.680	0.582	0.575

TABLE 5
Mixed derivative, 3600 *unknowns,* $m = 1$, *iterations* $= 53$, *relative time and speedup.*

	Number of processors				
Method	1	2	4	8	16
CW	0.961	0.967	0.967	0.978	0.971
	1.00	1.88	3.75	6.95	13.17
BD	0.838	0.849	0.848	0.874	0.871
	1.00	1.87	3.73	6.78	12.80
E	0.761	0.763	0.760	0.771	0.774
	1.00	1.89	3.78	6.98	13.08

TABLE 6
Mixed derivative, 3600 *unknowns,* $m = 2$, *iterations* $= 38$, *relative time and speedup.*

	Number of processors				
Method	1	2	4	8	16
CW	0.848	0.854	0.857	0.858	0.857
	1.00	1.90	3.78	6.94	12.99
BD	0.973	0.974	0.979	0.989	0.995
	1.00	1.91	3.79	6.90	12.83
BDE	0.829	0.849	0.852	0.853	0.849
	1.00	1.87	3.72	6.82	12.82
E	0.790	0.803	0.805	0.812	0.817
	1.00	1.88	3.75	6.83	12.70

TABLE 7
Mixed derivative, 3600 *unknowns,* $m = 4$, *iterations* $= 27$, *relative time and speedup.*

	Number of processors				
Method	1	2	4	8	16
CW	0.850	0.853	0.855	0.854	0.850
	1.00	1.92	3.82	6.97	12.97
BD	1.017	1.031	1.030	1.035	1.040
	1.00	1.90	3.79	6.88	12.68
BDE	0.908	0.943	0.944	0.944	0.935
	1.00	1.86	3.69	6.73	12.60
E	0.852	0.861	0.865	0.868	0.874
	1.00	1.91	3.78	6.87	12.64

TABLE 8
Plane stress, 2048 *unknowns,* $m = 1$, *iterations* = 88, *relative time and speedup.*

	Number of processors				
Method	1	2	4	8	16
CW	0.827	0.826	0.845	0.850	0.866
	1.00	1.93	3.80	7.49	14.12
BD	0.657	0.655	0.669	0.677	0.684
	1.00	1.93	3.82	7.47	14.21
E	0.623	0.626	0.638	0.640	0.648
	1.00	1.92	3.79	7.49	14.20

TABLE 9
Plane stress, 2048 *unknowns,* $m = 2$, *iterations* = 63, *relative time and speedup.*

	Number of processors				
Method	1	2	4	8	16
CW	0.819	0.819	0.838	0.841	0.857
	1.00	1.93	3.83	7.56	14.28
BD	0.923	0.913	0.937	0.944	0.950
	1.00	1.95	3.86	7.59	14.52
BDE	0.690	0.680	0.697	0.698	0.709
	1.00	1.96	3.87	7.66	14.54
E	0.668	0.675	0.689	0.696	0.706
	1.00	1.91	3.79	7.44	14.12

TABLE 10
Plane stress, 2048 *unknowns,* $m = 4$, *iterations* = 48, *relative time and speedup.*

	Number of processors				
Method	1	2	4	8	16
CW	0.814	0.819	0.836	0.838	0.851
	1.00	1.93	3.84	7.60	14.39
BD	0.949	0.946	0.971	0.979	0.985
	1.00	1.95	3.86	7.58	14.51
BDE	0.699	0.696	0.714	0.715	0.729
	1.00	1.95	3.86	7.64	14.44
E	0.681	0.691	0.704	0.715	0.723
	1.00	1.92	3.82	7.45	14.19

Tables 5–10 give parallel results for $m=1$, 2, and 4. Note that Tables 5 and 8 contain no listing for the BDE procedure since this only applies when $m>1$. Again, the times are relative to the straightforward implementation, and we have only used the smallest problem sizes: 3600 and 2048 unknowns. Note that the single processor times do not agree with those in Tables 3 and 4, even though the single processor codes have been stripped of all parallel overhead. This is due to the savings that the multicolor orderings allow in the straightforward implementation, as discussed in the previous section. Note also that the iterations given in Tables 5–10 are higher than for the corresponding problems in Tables 3 and 4. This is due to the multicoloring order; there is a trade-off between the parallelism of multicolor orderings and the degradation in the rate of convergence.

The parallel properties of the different procedures are roughly the same, as shown by the speedups. There are variations in these speedups, however, and no one procedure is uniformly superior. The E procedure has the best times in most cases although the BDE procedure is quite close for the plane stress problem. For the mixed derivative problem, however, the CW procedure becomes the best for $m=4$. This is due to the multicolor ordering and the resulting small savings on vector operations in the CW procedure, as mentioned in the previous section. For the mixed derivative problem, the cost of multiplication by L or L^T is so small that these savings become noticeable. We also note that for $m>1$, the BD procedure is sometimes worse than the straightforward approach. This is primarily due to the reduced operation count for SF using multicoloring.

Additional numerical results are given in Vaughan (1988) for larger sizes of the plane stress and mixed derivative problems, as well as some other problems from structural analysis. Experiments on a 64 processor Intel iPSC/1 hypercube for these problems are also reported. In all cases, the results are similar to those presented here.

9. Conclusions.

We have analyzed the CW and E procedures, and four different versions of the BD procedure. On the basis of the operation counts (Table 1) and storage (Table 2), we conclude that the E procedure is the most efficient for either one-step or m-step preconditioning. Our limited numerical experiments substantiate this conclusion except in a few cases where the CW procedure is best, due to the use of multicolor orderings on a problem with a very sparse matrix.

REFERENCES

L. ADAMS (1982), *Iterative algorithms for large sparse systems on parallel computers*, Ph.D. thesis, Department of Applied Mathematics, University of Virginia, Charlottesville, VA.

——— (1985), *M-step preconditioned conjugate gradient methods*, SIAM J. Sci. Statist. Comput., 6, pp. 452–463.

R. BANK AND C. DOUGLAS (1985), *An efficient implementation for* SSOR *and incomplete factorization preconditionings*, Appl. Numer. Math., 1, pp. 489–492.

V. CONRAD AND Y. WALLACH (1979), *Alternating methods for sets of linear equations*, Numer. Math., 27, pp. 371–372.

S. EISENSTAT (1981), *Efficient implementation of a class of conjugate gradient methods*, SIAM J. Sci. Statist. Comput., 2, pp. 1–4.

O. JOHNSON, C. MICCHELI, AND G. PAUL (1983), *Polynomial preconditioners for conjugate gradient calculation*, SIAM J. Numer. Anal., 20, pp. 362–376.

J. ORTEGA (1988a), *Efficient implementation of certain iterative methods*, SIAM J. Sci. Statist. Comput., 9, pp. 882–891.

——— (1988b), *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum Press, New York.

C. VAUGHAN (1988), *The* SSOR *preconditioned conjugate gradient method on parallel computers*, Ph.D. thesis, Department of Applied Mathematics, University of Virginia, Charlottesville, VA.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 5, pp. 873-912, September 1990

ACCURATE SINGULAR VALUES OF BIDIAGONAL MATRICES*

JAMES DEMMEL† AND W. KAHAN‡

Abstract. Computing the singular values of a bidiagonal matrix is the final phase of the standard algorithm for the singular value decomposition of a general matrix. A new algorithm that computes all the singular values of a bidiagonal matrix to high relative accuracy independent of their magnitudes is presented. In contrast, the standard algorithm for bidiagonal matrices may compute small singular values with no relative accuracy at all. Numerical experiments show that the new algorithm is comparable in speed to the standard algorithm, and frequently faster.

Key words. singular value decomposition, bidiagonal matrix, QR iteration

AMS(MOS) subject classifications. 65F20, 65G05, 65F35

1. Introduction.

The standard algorithm for computing the singular value decomposition (SVD) of a general real matrix A has two phases [7]:

(1) Compute orthogonal matrices P_1 and Q_1 such that $B = P_1^T A Q_1$ is in bidiagonal form, i.e., has nonzero entries only on its diagonal and first superdiagonal.

(2) Compute orthogonal matrices P_2 and Q_2 such that $\Sigma = P_2^T B Q_2$ is diagonal and nonnegative. The diagonal entries σ_i of Σ are the *singular values* of A. We will take them to be sorted in decreasing order: $\sigma_i \geqq \sigma_{i+1}$. The columns of $Q = Q_1 Q_2$ are the *right singular vectors*, and the columns of $P = P_1 P_2$ are the *left singular vectors.*

This process takes $O(n^3)$ operations, where n is the dimension of A. This is true even though Phase 2 is iterative, since it converges quickly in practice. The error analysis of this combined procedure has a widely accepted conclusion [8], and provided neither overflow nor underflow occurs may be summarized as follows:

The computed singular values σ_i differ from the true singular values of A by no more than $p(n) \cdot \varepsilon \cdot \|A\|$, where $\|A\| = \sigma_1$ is the 2-norm of A, ε is the machine precision, and $p(n)$ is a slowly growing function of the dimension n of A.

This is a generally satisfactory conclusion, since it means the computed singular values have errors no larger than the uncertainty in the largest entries of A, if these are themselves the results of previous computations. In particular, singular values not much smaller than $\|A\|$ are computable to high relative accuracy. However, small singular values may change completely, and so cannot generally be computed with high relative accuracy.

There are some situations where the smallest singular values are determined much more accurately by the data than a simple bound of the form $p(n)\varepsilon\|A\|$ would indicate. In this paper we will show that for bidiagonal matrices the singular values are determined to the same relative precision as the individual matrix entries. In other words, if all the matrix entries are known to high relative accuracy, all the singular values are also known to high relative accuracy independent of their magnitudes. This will follow from an analogous theorem about the eigenvalues of symmetric tridiagonal matrices with zero diagonal.

* Received by the editors July 18, 1988; accepted for publication (in revised form) July 12, 1989.

† Courant Institute of Mathematical Sciences, New York University, New York, New York 10012. This author was supported as a Presidential Young Investigator and under National Science Foundation grant ASC-8715728.

‡ Computer Science Division, University of California, Berkeley, California 94720. The work of this author was partially supported by grants from the Research Offices of the U.S. Army, Navy, and Air Force under contracts numbered, respectively, DAA-629-85-K-0070, N00014-85-K-0013, and AFOSR-84-0158.

In such situations it is desirable to have an algorithm to compute the singular values or eigenvalues to the accuracy to which they are determined by the data. In this paper we present an algorithm for computing all the singular values of a bidiagonal matrix to *guaranteed* high relative accuracy, independent of their magnitudes. Our algorithm is a variation of the usual QR iteration that is used in the standard SVD algorithm. Briefly, it is a hybrid algorithm of the usual QR iteration with a "zero-shifted" QR modified to guarantee forward stability. Numerical experience, which we report below, shows that it is generally faster than the standard algorithm, and ranges from 2.7 times faster to 1.6 times slower counting reduction to bidiagonal form (7.7 times faster to 3.4 times slower not counting reduction to bidiagonal form).

This perturbation theory and algorithm also apply to some classes of symmetric matrices. For example, they may be applied to symmetric tridiagonal matrices with zero diagonal; such matrices arise by reducing skew-symmetric matrices to tridiagonal form. Another class where the perturbation theory applies, so that small relative perturbations in the matrix entries only cause small relative perturbations in the eigenvalues, are scaled diagonally dominant symmetric matrices. A symmetric matrix H is scaled diagonally dominant if $H = DAD$, where D is an arbitrary diagonal matrix and A is symmetric and diagonally dominant in the usual sense. This class includes all symmetric positive definite matrices that may be consistently ordered [1], a class that arises in the numerical solution of elliptic partial differential equations. In particular, this class includes all symmetric positive-definite tridiagonal matrices. As before, we can exhibit algorithms to compute the eigenvalues of H to their inherent accuracy. This work will be reported on elsewhere [1].

The rest of this paper is organized as follows. Section 2 presents perturbation theory for the singular values of a bidiagonal matrix, and shows that small relative perturbations in the nonzero entries of a bidiagonal matrix can only cause small relative perturbations in its singular values. We also present theorems that say when an off-diagonal entry can be set to zero without making large relative perturbations in any singular value; these theorems are the basis of the convergence criteria for the new algorithm. Section 3 presents the algorithm, which is QR iteration with a "zero shift," modified to be forward stable. This forward stability combined with the perturbation theorem of § 2 shows that QR can compute all the singular values with high relative accuracy. Section 4 discusses convergence criteria for the new algorithm, since the convergence criteria for the standard algorithm can cause unacceptably large perturbations in small singular values. It also discusses the practical algorithm, which is a hybrid of the standard algorithm and the algorithm of § 3. Details of the implementation, including high-level code for the entire algorithm, are presented in § 5. Sections 3, 4, and 5 may be read independently of § 2. Section 6 shows how to use bisection, Rayleigh quotient iteration, and various other schemes to compute the singular values of a bidiagonal matrix to high relative accuracy. Bisection will be used to verify the results in § 7, which discusses numerical experiments. Section 7 also addresses the implications of our results for the "perfect shift" strategy for computing singular vectors. Section 8 contains a detailed error analysis of the new algorithm. Section 9 discusses the accuracy of the computed singular vectors; a complete analysis of this remains an open question. Sections 6–9 may be read independently. Sections 7 and 8 depend only on §§ 3–5. Section 10 contains suggestions for parallel versions of the algorithms presented, open questions, and conclusions.

2. Perturbation theory for singular values of bidiagonal matrices.

We say δa is a relative perturbation of a of size at most η if $|\delta a| \leqq \eta |a|$. If A and δA are matrices, we will let $|A|$ and $|\delta A|$ denote the matrices of absolute entries of A and δA. We will say that δA is a componentwise relative perturbation of A of size at most η if $|\delta A| \leqq \eta |A|$, where the inequality is understood componentwise.

In this section we will prove three perturbation theorems for singular values of bidiagonal matrices. The first theorem is needed to prove that our new QR iteration does not disturb any singular values, and the second two theorems justify our new convergence criteria (see § 4 below).

The first theorem shows that if δB is a componentwise relative perturbation of size η of the n-by-n bidiagonal matrix B, then the singular values σ_i' of $B+\delta B$ will be relative perturbations of the singular values σ_i of B of size less than about $(2n-1)\eta$, provided $(2n-1)\eta$ is small compared to 1. More precisely we will show that

$$(1-\eta)^{2n-1} \cdot \sigma_i \leqq \sigma_i' \leqq (1-\eta)^{1-2n} \cdot \sigma_i$$

(recall that σ_i' and σ_i are sorted in decreasing order). This will follow as a corollary of a more general result for symmetric tridiagonal matrices with zero diagonal.

The last two theorems say when we can set an off-diagonal entry of a bidiagonal matrix B to zero without making large relative perturbations in the singular values. They are based on a simple recurrence for estimating the smallest singular value of a bidiagonal matrix; if setting an off-diagonal entry of B to zero cannot change this recurrence significantly, we show that no singular value can be changed significantly either.

The proof of the first theorem depends on Sylvester's Law Of Inertia [6, p. 297]:

SYLVESTER'S LAW OF INERTIA. Let A be symmetric and U be nonsingular. Then A and UAU^T have the same number of positive, zero, and negative eigenvalues.

In particular, suppose A is symmetric and tridiagonal, with diagonal entries $a_1, \cdots, a_n$ and off-diagonal entries $b_1, \cdots, b_{n-1}$. Then via Gaussian elimination without pivoting we can write $A - xI = LDL^T$, where L is unit lower triangular and bidiagonal, and D is diagonal with entries d_i given by the recurrence [15, p. 47]

$$\begin{aligned} d_1 &= a_1 - x, \\ d_i &= a_i - x - b_{i-1}^2 / d_{i-1}. \end{aligned} \tag{2.1}$$

This recurrence will not break down ($d_i = 0$ for some $i < n$) as long as x is not one of the $n(n-1)/2$ eigenvalues of leading submatrices of A. Then by Sylvester's Law of Inertia, the numbers of eigenvalues of A less than x, equal to x, and greater than x are precisely the numbers of d_i that are negative, zero, and positive, respectively.

We will also need the following classical eigenvalue perturbation theorem due to Weyl.

THEOREM 1 [15, p. 191]. *Let $\lambda_1 \geqq \cdots \geqq \lambda_n$ be the eigenvalues of the symmetric matrix A, and $\lambda_1' \geqq \cdots \geqq \lambda_n'$ be the eigenvalues of the symmetric matrix $A+\delta A$. Then $-\|\delta A\| \leqq \lambda_{\min}(\delta A) \leqq \lambda_i' - \lambda_i \leqq \lambda_{\max}(\delta A) \leqq \|\delta A\|$. Here, $\lambda_{\min}$ and $\lambda_{\max}$ denote the smallest and largest eigenvalues, respectively.*

Now we present our central result of this section (a slightly weaker version originally appeared in an unpublished report [12]).

THEOREM 2. *Let J be an n-by-n symmetric tridiagonal matrix with zero diagonal and off-diagonal entries $b_1, \cdots, b_{n-1}$. Suppose $J+\delta J$ is identical to J except for one off-diagonal entry, which changes to αb_i from b_i, $\alpha \neq 0$. Let $\bar{\alpha} = \max(|\alpha|, |\alpha^{-1}|)$. Let λ_i be the eigenvalues of J sorted into decreasing order, and let λ_j' be the eigenvalues of $J+\delta J$ similarly sorted. Then*

$$\frac{\lambda_i}{\bar{\alpha}} \leqq \lambda_i' \leqq \bar{\alpha} \cdot \lambda_i. \tag{2.2}$$

In other words, changing any single entry of J by a factor α can change no eigenvalue by more than a factor $|\alpha|$.

Proof. Assume without loss of generality that $\alpha > 0$, and no b_i is zero, since otherwise J is block diagonal, and each diagonal block may be analyzed separately. The recurrences corresponding to (2.1) for $J - xI$ and $J + \delta J - xI$ may be written as follows:

$$\begin{aligned} u_1 &= -x, \\ u_{k+1} &= -x - b_k^2/u_k, \end{aligned} \qquad \text{and} \qquad \begin{aligned} v_1 &= -x, \\ v_{k+1} &= -x - b_k^2/v_k, \ k \neq i, \\ v_{i+1} &= -x - \alpha^2 b_i^2/v_i. \end{aligned}$$

Since both J and $J + \delta J$ have nonzero off-diagonals, they must have simple eigenvalues [15, p. 124] λ_i and λ_i', respectively. As long as x is not one of the $n(n-1)$ eigenvalues of leading principal submatrices of J and $J + \delta J$, no division by zero will occur in these recurrences. Also, $u_n = 0$ if and only if x is an eigenvalue of J, and $v_n = 0$ if and only if x is an eigenvalue of $J + \delta J$.

Our goal is to show that each λ_i is the ith eigenvalue of some symmetric matrix $J(\lambda_i)$ that differs from $J + \delta J$ by a matrix $X = J + \delta J - J(\lambda_i)$ satisfying

$$\text{(2.3)} \qquad \begin{aligned} (\bar\alpha^{-1} - 1)\lambda_i \leqq \lambda_{\min}(X) \leqq \lambda_{\max}(X) \leqq (\bar\alpha - 1)\lambda_i \quad &\text{if } \lambda_i \geqq 0, \\ (\bar\alpha - 1)\lambda_i \leqq \lambda_{\min}(X) \leqq \lambda_{\max}(X) \leqq (\bar\alpha^{-1} - 1)\lambda_i \quad &\text{if } \lambda_i < 0; \end{aligned}$$

together with Theorem 1 these inequalities will yield the desired result.

We construct $J(\lambda_i)$ as follows. Let

$$\begin{aligned} w_j &= u_j \cdot \alpha^{(-1)^{i-j}} \quad \text{if } j \leqq i, \\ w_j &= u_j \cdot \alpha^{(-1)^{i-j-1}} \quad \text{if } j > i. \end{aligned}$$

Note that the w_j satisfy the recurrence

$$\begin{aligned} w_1 &= -x\alpha^{(-1)^{i-1}}, \\ w_{j+1} &= -x\alpha^{(-1)^{i-j-1}} - b_j^2/w_j \quad \text{if } j < i, \\ w_{i+1} &= -x\alpha - \alpha^2 b_i^2/w_i, \\ w_{j+1} &= -x\alpha^{(-1)^{i-j}} - b_j^2/w_j \quad \text{if } j > i, \end{aligned}$$

or

$$\begin{aligned} w_1 &= -x - x_1, \\ w_{j+1} &= -x - x_{j+1} - b_j^2/w_j \quad \text{if } j < i, \\ w_{i+1} &= -x - x_{i+1} - \alpha^2 b_i^2/w_i, \\ w_{j+1} &= -x - x_{j+1} - b_j^2/w_j \quad \text{if } j > i, \end{aligned}$$

which is the recurrence for $J(x) = J + \delta J - X$ where $X = \operatorname{diag}(x_i)$, $x_i = (\alpha^{\pm 1} - 1)x$. Now set $x = \lambda_i$. Since the w_j and u_j sequences have the same signs by construction (including $u_n = w_n = 0$), λ_i is the ith eigenvalue of $J(\lambda_i)$. Furthermore, $\lambda_{\max}(X)$ and $\lambda_{\min}(X)$ clearly satisfy (2.3) above. □

As an immediate corollary we get the following corollary.

COROLLARY 1. *Let J be an n-by-n symmetric tridiagonal matrix with zero diagonal and off-diagonal entries $b_1, \cdots, b_{n-1}$. Let $J + \delta J$ have off-diagonal entries*

$\alpha_1 b_1, \cdots, \alpha_{n-1} b_{n-1}$, $\alpha_i \neq 0$. Let $\bar{\alpha} = \prod_{i=1}^{n-1} \max(|\alpha_i|, |\alpha_i^{-1}|)$. Let λ_i be the eigenvalues of J sorted into decreasing order, and λ_i' be the eigenvalues of $J + \delta J$ similarly sorted. Then

$$\frac{\lambda_i}{\bar{\alpha}} \leqq \lambda_i' \leqq \bar{\alpha} \cdot \lambda_i.$$

For example, if $1 - \eta \leqq |\alpha| \leqq 1 + \eta$, no eigenvalue can change by a factor exceeding $\bar{\alpha} = (1-\eta)^{-n+1}$.

We can apply Theorem 2 to prove a similar theorem for singular values of bidiagonal matrices by noting that for any matrix B the eigenvalues of

$$B' \equiv \begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix}$$

are the singular values of B, their negatives, and some zeros (if B is not square) [8, p. 286]. Suppose now that B is n by n and bidiagonal with diagonal entries $s_1, \cdots, s_n$ and superdiagonal entries $e_1, \cdots, e_{n-1}$. Then by permuting the rows and columns of B' to appear in the new order $1, n+1, 2, n+2, \cdots, n, 2n$, we see B' is orthogonally similar to the tridiagonal matrix B'' with zeros on the diagonal and off-diagonals $s_1, e_1, s_2, e_2, \cdots, e_{m-1}, s_m$ [7, p. 213]. Thus the singular values of B are the absolute values of the eigenvalues of the matrix B'', which is of the form required by Theorem 2. This proves Corollary 2.

COROLLARY 2. *Let B be an n-by-n bidiagonal matrix and suppose $\delta B_{ii} + B_{ii} = \alpha_{2i-1} B_{ii}$, $\delta B_{i,i+1} + B_{i,i+1} = \alpha_{2i} B_{i,i+1}$, $\alpha_j \neq 0$. Let $\bar{\alpha} = \prod_{i-1}^{2n-1} \max(|\alpha_i|, |\alpha_i^{-1}|)$. Let $\sigma_1 \geqq \cdots \geqq \sigma_n$ be the singular values of B, and let $\sigma_1' \geqq \cdots \geqq \sigma_n'$ be the singular values of $B + \delta B$. Then*

$$\frac{\sigma_i}{\bar{\alpha}} \leqq \sigma_i' \leqq \bar{\alpha} \cdot \sigma_i.$$

For example, if $1 - \eta \leqq |\alpha_j| \leqq 1 + \eta$, then no singular value can change by more than a factor of $\bar{\alpha} = (1-\eta)^{1-2n}$.

That this result is essentially best possible may be seen by considering the n-by-n matrix

$$B(\eta) = \begin{bmatrix} 1-\eta & \beta(1+\eta) & \cdot_{\cdot_{\cdot}} & \\ & \ddots & & \\ & & \ddots & \beta(1+\eta) \\ & & & 1-\eta \end{bmatrix}.$$

When $\beta \gg 1$, the smallest singular value is approximately $\beta^{1-n}(1-(2n-1)\eta)$.

This theorem may be contrasted with the following classical perturbation bound for singular values, where it is only possible to bound the absolute perturbation in the singular values of a perturbed general matrix.

THEOREM 3 [8, p. 286]. *Let $\sigma_1 \geqq \cdots \geqq \sigma_n$ be the singular values of A, and $\sigma_1' \geqq \cdots \geqq \sigma_n'$ be the singular values of $A + \delta A$. Then $|\sigma_i' - \sigma_i| \leqq \|\delta A\|$.*

One caveat about the use of Corollary 2 in practice is that phase 1 of the SVD algorithm, reduction to bidiagonal form, may produce completely inaccurate bidiagonal entries. Sometimes, however, the reduction to bidiagonal form is quite accurate, so that the singular values of the original matrix can be computed accurately (see [1] for discussion).

In § 6 we will show how to use recurrence (2.1) in practice to compute the singular values of a bidiagonal matrix with guaranteed high relative accuracy. This method, although not competitive in speed on a serial machine with the algorithm of the next section, can be used to efficiently verify the accuracy of the singular values computed by another method. The algorithm based on (2.1) may also be parallelized easily (see § 6).

The second result of this section tells us when we can set an off-diagonal of B to zero without making large relative changes in the singular values. This theorem will justify one of the convergence criteria we describe in § 4 below.

First we discuss a simple recurrence for approximating the smallest singular value of a bidiagonal matrix, which also appeared in [9].

LEMMA 1. *Let B be an n-by-n bidiagonal matrix with nonzero diagonal entries $s_1, \cdots, s_n$ and nonzero off-diagonal entries $e_i, \cdots, e_{n-1}$. Consider the following recurrences*:

(2.4)
```
$\lambda_n = |s_n|$
for $j = n-1$ to 1 step $-1$ do
    $\lambda_j = |s_j| \cdot (\lambda_{j+1}/(\lambda_{j+1}+|e_j|))$
$\mu_1 = |s_1|$
for $j = 1$ to $n-1$ do
    $\mu_{j+1} = |s_{j+1}| \cdot (\mu_j/(\mu_j+|e_j|))$.
```

Then $\|B^{-1}\|_\infty^{-1} = \min_j \lambda_j$ and $\|B^{-1}\|_1^{-1} = \min_j \mu_j$. Furthermore, letting $\underline{\sigma} \equiv \min(\|B^{-1}\|_1^{-1}, \|B^{-1}\|_\infty^{-1})$, we have

$$\begin{gathered} n^{-1/2} \cdot \|B^{-1}\|_\infty^{-1} \leqq \sigma_{\min}(B) \leqq n^{1/2} \cdot \|B^{-1}\|_\infty^{-1}, \\ n^{-1/2} \cdot \|B^{-1}\|_1^{-1} \leqq \sigma_{\min}(B) \leqq n^{1/2} \cdot \|B^{-1}\|_1^{-1}, \\ \underline{\sigma} \leqq \sigma_{\min}(B) \leqq n^{1/2} \cdot \underline{\sigma}. \end{gathered} \tag{2.5}$$

Proof. By means of pre and postmultiplication by unitary diagonal matrices with diagonal entries of unit modulus, we may assume that $s_i > 0$ and $e_i < 0$. Then B^{-1} is easily seen to have positive superdiagonal entries, so that $\|B^{-1}\|_\infty = \|B^{-1}u\|_\infty$ and $\|B^{-1}\|_1 = \|u^TB^{-1}\|_1$, where u is the vector of all ones. $v = B^{-1}u$ and $w^T = u^TB^{-1}$ are easily computed by back and forward substitution. Thus $\|B^{-1}\|_\infty = \max_i |v_i|$ and $\|B^{-1}\|_1 = \max_i |w_i|$. Modifying these back and forward substitution recurrences to compute $\lambda_i = 1/v_i$ and $\mu_i = 1/w_i$ yields the recurrences in (2.4). Since the eigenvalues of

$$H = \begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix}$$

are the positive and negative singular values of B,

$$\|B^{-1}\| = \|H^{-1}\| \leqq \|H^{-1}\|_\infty = \max(\|B^{-1}\|_\infty, \|B^{-T}\|_\infty) = \max(\|B^{-1}\|_\infty, \|B^{-1}\|_1),$$

proving the inequality $\underline{\sigma} \leqq \sigma_{\min}(B)$ in (2.5). The other inequalities are standard norm inequalities. □

From Lemma 1 it is clear that if $|e_j/\lambda_{j+1}| \leqq \eta < 1$, then changing e_j to 0 can make a relative change of at most η in λ_j and all subsequent λ_i, $i<j$. Thus the first upper and lower bounds on $\sigma_{\min}(B)$ in (2.5) can change only by a factor of η as well. Similar comments apply if $|e_j/\mu_j| \leqq \eta < 1$. This suggests the following criterion for setting e_j to 0:

Convergence Criterion 1. Let $\eta < 1$ be the desired relative accuracy of computed singular values. Then if either $|e_j/\lambda_{j+1}| \leqq \eta$ or $|e_j/\mu_j| \leqq \eta$, set e_j to 0.

Now we will state and prove a theorem that justifies this criterion. We will only prove the theorem for the case $|e_j/\lambda_{j+1}| \leqq \eta$; the case $|e_j/\mu_j| \leqq \eta$ is analogous. First we need some notation. Let $\phi(\eta)$ be the unique positive solution of

$$\exp(\phi) - 2\phi - 1 = \eta^2; \tag{2.6}$$

it is easy to see that $\phi(\eta)$ is asymptotically $\eta/2^{1/2}$ for small η and that for all η, $\phi(\eta) \leqq \eta/2^{1/2}$. Let B be a bidiagonal matrix as in Lemma 1 with singular values $\sigma_1 \geqq \cdots \geqq \sigma_n$, let $U = B$ except for entry e_j which is zero, and let $\sigma'_1 \geqq \cdots \geqq \sigma'_n$ be the singular values of U. Let $\mathcal{I}_i(\eta)$ be the interval of σ's such that

$$-\phi(\eta) \leqq \ln(\sigma/\sigma'_i) \leqq \phi(\eta); \tag{2.7}$$

$\mathcal{I}_i(\eta)$ is essentially the set of σ that differs from σ'_i by a relative perturbation of at most $\eta/2^{1/2}$. Some of these intervals may overlap; let $\mathcal{C}(\eta)$ denote the collection of disjoint intervals made of connected components of $\bigcup_i \mathcal{I}_i(\eta)$. Now we may state Theorem 4.

THEOREM 4. *Let B and U be bidiagonal matrices as described above, and suppose $|e_j/\lambda_{j+1}| \leqq \eta$. Then each singular value σ_i of B lies in the connected component of $\mathcal{C}(\eta)$ containing $\mathcal{I}_i(\eta)$. In particular, if that connected component consists of m overlapping intervals $\mathcal{I}_j(\eta)$, then*

$$-m\phi(\eta) \leqq \ln(\sigma'_i/\sigma_i) \leqq m\phi(\eta). \tag{2.8}$$

Therefore, the relative perturbation caused in σ_i by setting the off-diagonal entry in δB to zero is at most $n\eta/2^{1/2}$ if $n\eta \ll 1$, and if σ_i is sufficiently separated from the other singular values, at most $\eta/2^{1/2}$.

For example, this theorem lets us conclude that setting η to 0 in

$$\begin{bmatrix} 1 & \eta & \\ & 1 & 1 \\ & & \alpha \end{bmatrix}$$

can change the singular values $2^{1/2}$, 1, and $2^{-1/2}\alpha$ by at most factors of $1 \pm \eta$ if η is small, independent of α. Indeed, if D is *any* bidiagonal matrix this theorem guarantees that we can set η to 0 in

$$\begin{bmatrix} 1 & \eta \\ & D \end{bmatrix}$$

without making relative perturbations larger than η in *any* singular value.

The proof of this theorem will depend on a sequence of technical lemmas. The first is a trivial consequence of Taylor's theorem.

LEMMA 2. *Let f and g be continuously differentiable functions on the non-negative real axis, with $f(t) < g(t)$ for t positive and sufficiently small. Let $\xi \equiv \inf\{t > 0 : f(t) \geqq g(t)\}$, and $\xi = \infty$ if no such t exist. Then if ξ is finite, $f'(\xi) \geqq g'(\xi)$.*

We will use contrapositive of this result to show when $f < g$ for all t; if $f(\xi) \geqq g(\xi)$ would imply that $f'(\xi) < g'(\xi)$, then f must be less than g everywhere.

In our case, we define $f(t)$ and $g(t)$ as follows. Write

$$B = \begin{bmatrix} K & C \\ & R \end{bmatrix},$$

where K is j by j, R is $n-j$ by $n-j$, and $C = e_j l f^T$, where $l = (0, \cdots, 0, 1)^T$ and $f = (1, 0, \cdots, 0)^T$. Assume as in Lemma 1 that $e_j < 0$. Let

$$U(t) = \begin{bmatrix} K & C(t) \\ & R \end{bmatrix},$$

where $C(t) = -t\lambda_{j+1} lf^T$ so that $U(0) = U$ and $U(\eta) = B$. Let $\sigma_i'(t)$ be the ith largest singular value of $U(t)$. Then we first let

$$f(t) = -\phi(t) \quad \text{and} \quad g(t) = \ln(\sigma_i'(t)/\sigma_i') \tag{2.9a}$$

and apply Lemma 2 to prove the first inequality in (2.8), and then let

$$f(t) = \ln(\sigma_i'(t)/\sigma_i') \quad \text{and} \quad g(t) = \phi(t) \tag{2.9b}$$

to prove the second inequality. In order to apply Lemma 2 to draw this conclusion, we need to compute the derivatives of the functions in (2.9a), (2.9b).

LEMMA 3. *Unless $\sigma_i'(t)$ is a singular value of R,*

$$\min\left(\min_j \frac{t}{((\sigma_i'(t)/\sigma_j')^2 - 1)}, 0\right) < \frac{d}{dt}\ln(\sigma_i'(t)) < \max\left(\max_j \frac{t}{((\sigma_i'(t)/\sigma_j')^2 - 1)}, 0\right).$$

Proof. We begin with a simplifying assumption. We assume K and R have no common singular values. If this is not true, consider a sequence of problems with $K_n \to K$, $R_n \to R$ and where K_n and R_n have distinct singular values; the general result will follow from continuity.

We may define a singular value $\sigma(t)$ and its singular vectors $u(t)$ and $v(t)$ of $U(t)$ by the equations $Uv = \sigma u$ and $u^TU = \sigma v^T$ (where we have suppressed the argument t). Using the fact that $u^Tu = v^Tv > 0$, we see from $\dot{U}v + U\dot{v} = \dot{\sigma}u + \sigma\dot{u}$ and from $\dot{u}^TU + u^T\dot{U} = \dot{\sigma}v^T + \sigma\dot{v}^T$ that

$$\dot{\sigma} = u^T\dot{U}v/v^Tv = u^T\dot{U}v/u^Tu.$$

Now partition $u^T = (u_1^T, u_2^T)$ and $v^T = (v_1^T, v_2^T)$ conformally to B, whence

$$\begin{matrix} Kv_1 + Cv_2 = \sigma u_1, \\ Rv_2 = \sigma u_2, \end{matrix} \quad \text{and} \quad \begin{matrix} u_1^TK = \sigma v_1^T, \\ u_1^TC + u_2^TR = \sigma v_2^T. \end{matrix} \tag{2.10}$$

Now

$$\dot{U}(t) = \begin{bmatrix} 0 & \dot{C} \\ 0 & 0 \end{bmatrix} \quad \text{where} \quad \dot{C} = -\lambda_{j+1} \cdot lf^T.$$

By rearranging the recurrence (2.4) for λ_{j+1} we see that $\lambda_{j+1} = \|R^{-T}f\|_1^{-1}$. Thus

$$\dot{\sigma}(t) = \frac{u_1^T\dot{C}v_2}{u_1^Tu_1 + u_2^Tu_2} = \frac{-u_1^Tlf^Tv_2}{\|R^{-T}f\|_1(u_1^Tu_1 + u_2^Tu_2)}. \tag{2.11}$$

Now we derive another expression for v_2 in order to eliminate it from (2.11). Since

$$R^TRv_2 = \sigma R^Tu_2 = \sigma^2v_2 - \sigma C^Tu_1 = \sigma^2v_2 + \sigma tfl^Tu_1\|R^{-T}f\|_1^{-1},$$

we may solve for v_2 as follows provided σ is not a singular value of R:

$$v_2 = \sigma tl^Tu_1(R^TR - \sigma^2I)^{-1}f\|R^{-T}f\|_1^{-1} = \sigma tl^Tu_1R^{-1}(I - \sigma^2(R^TR)^{-1})^{-1}R^{-T}f\|R^{-T}f\|_1^{-1}$$

and so

$$\begin{aligned} \frac{d}{dt}\ln(\sigma(t)) &= \dot{\sigma}(t)/\sigma(t) \\ &= t \cdot \frac{(l^Tu_1)^2}{u_1^Tu_1 + u_2^Tu_2} \cdot \frac{\|R^{-T}f\|^2}{\|R^{-T}f\|_1^2} \cdot \frac{(R^{-T}f)^T(\sigma^2(R^TR)^{-1} - I)^{-1}R^{-T}f}{\|R^{-T}f\|^2}. \end{aligned} \tag{2.12}$$

Since l is a unit vector, the second factor in this expression is between 0 and 1. It cannot be zero because otherwise $C^Tu_1 = 0$, $Rv_2 = \sigma u_2$, and $u_2^TR = \sigma v_2^T$, and so σ would be a singular value of R contrary to assumption. The third factor is strictly between 0 and 1. The last factor is a Rayleigh quotient and so bounded by the extreme eigenvalues of the matrix in the middle, i.e., $\min_j ((\sigma/\sigma_{Rj})^2-1)^{-1}$, and $\max_j ((\sigma/\sigma_{Rj})^2-1)^{-1}$, where σ_{Rj} are the singular values of R. This is in turn bounded by the extreme values of $1/((\sigma/\sigma'_j)^2-1)$. This proves the lemma. □

LEMMA 4. $\phi(0)=0$ *and* $\dot\phi(0)=2^{-1/2}$. $\phi(t)$ *satisfies the differential equation*

$$\dot\phi(t)=\frac{t}{\exp(2\phi(t))-1} \tag{2.13a}$$

and $\psi(t)=-\phi(t)$ *satisfies the differential equation*

$$\dot\psi(t)=\frac{t}{1-\exp(-2\psi(t))}. \tag{2.13b}$$

Proof. Simply differentiate the defining equation (2.6) for $\phi(t)$. □

Proof of Theorem 4. Now note that $\ln(\sigma'_i(0)/\sigma'_i)=0$ and its derivative $\dot\sigma'_i(0)/\sigma'_i(0)=0$ as well since $\sigma'_i(t)$ is an even function of t. Since $\phi(0)=0$ and $\dot\phi(0)=2^{-1/2}$, we see that (2.8) is true (for $m=1$) for sufficiently small η. To show it is true for all η, we assume to the contrary that there is some positive η for which it is false, and let ξ be the infimum of all these η. Then $\sigma'_i(\xi)$ will be on the boundary of $\mathscr{C}(\eta)$, which means $|\ln(\sigma'_i(\xi)/\sigma'_j)|$ will be at least $\phi(\xi)$ for all j. From Lemma 3 we see this implies

$$\frac{-t}{1-\exp(-2\phi(\xi))}<\frac{d}{dt}\ln(\sigma'_i(\xi)/\sigma'_i)<\frac{t}{\exp(2\phi(\xi))-1}.$$

But we also have from Lemma 4 that

$$\dot\psi(\xi)=\frac{t}{1-\exp(-2\psi(\xi))} \quad\text{and}\quad \dot\phi(\xi)=\frac{t}{\exp(2\phi(\xi))-1}.$$

Therefore, the choice (2.9a) of f and g yields $\dot f(\xi)<\dot g(\xi)$, so $f(\xi)$ cannot equal $g(\xi)$. The choice (2.9b) yields the same conclusion. Therefore, $\sigma'_i(\xi)$ cannot lie on the boundary of $\mathscr{C}(\xi)$ as supposed. This completes the proof of Theorem 4. □

The third result in this section supplies a convergence criterion that may occasionally succeed in setting an off-diagonal entry to zero before Convergence Criterion 1. However, it may only be applied when singular vectors are *not* computed, since it may cause rather large perturbations in them. Let

$$B=\begin{bmatrix} D & \tilde e \\ 0 & s\end{bmatrix} \quad\text{and}\quad B'=\begin{bmatrix} D & 0 \\ 0 & s\end{bmatrix}, \tag{2.14}$$

where $\tilde e=[0,\cdots,0,e]^T$, and D is bidiagonal. Let $\sigma_1\leqq\cdots\leqq\sigma_n$ be the singular values of B and $\sigma'_1\leqq\cdots\leqq\sigma'_n$ be the singular values of B'. Let $\tilde{\mathscr{I}}_i(\eta)$ be the interval of σ's such that $|\sigma-\sigma'_i|\leqq\eta\sigma'_i$, and let $\tilde{\mathscr{C}}(\eta)$ be the collection of disjoint intervals that are the connected components of $\bigcup_i\tilde{\mathscr{I}}_i(\eta)$. Now we may state Theorem 5.

THEOREM 5. *Let* $0<\eta<1$ *be a relative error tolerance, and suppose* $gap\equiv\sigma_{\min}(D)-|s|>0$ *in* (2.14). *If*

$$|e|^2\leqq .5\cdot\eta\cdot gap\cdot(\sigma_{\min}(D)+|s|)=.5\cdot\eta\cdot(\sigma^2_{\min}(D)-|s|^2), \tag{2.15}$$

then each singular value σ'_i *of* B' *lies in the connected component of* $\tilde{\mathscr{C}}(\eta)$ *containing* $\tilde{\mathscr{I}}_i(\eta)$. *In particular, if that connected component consists of m overlapping intervals* $\tilde{\mathscr{I}}_j(\eta)$, *and* $m\eta\ll 1$, *then* $|\sigma'_i-\sigma_i|$ *is at most about* $m\cdot\eta\cdot\sigma'_i$.

Proof. We consider two cases: $se/[gap(\sigma_{\min}(D)+|s|)]\geqq .5$, and $se/[gap(\sigma_{\min}(D)+|s|)]<.5$. In the first case (2.15) implies $e^2<\eta se$ or $e<\eta s$. Then by Theorem 3 setting

e to 0 in B can change no singular value by more than $|\eta s|$, proving the theorem in this case.

Now consider the second case. Instead of directly comparing the singular values of B and B', we compare the eigenvalues of BB^T and $B'B'^T$, which are the squares of the corresponding singular values. First we show that the smallest eigenvalues of BB^T and $B'B'^T$ must be close. The smallest eigenvalue of $B'B'^T$ is s^2. By Theorem 1, BB^T has one eigenvalue less than $s^2+se \leqq (\sigma^2_{\min}(D)-s^2)/2$, and the rest exceeding the same quantity. By the gap theorem [15, § 11-7-1], the smallest eigenvalue σ^2_n of BB^T satisfies

$$|\sigma_n^2 - s^2| \leqq \frac{2s^2e^2}{\sigma^2_{\min}(D)-s^2} \leqq \eta s^2$$

proving the theorem for the smallest eigenvalue.

Now we consider the larger eigenvalues of BB^T. Since $se/[gap(\sigma_{\min}(D)-|s|)] < .5$, Theorem 4.12 of [16] applies and we conclude that the larger eigenvalues of BB^T are the same as the eigenvalues of $DD^T+E_1+E_2$, where

$$E_1 = \begin{bmatrix} 0 & \cdot & 0 \\ \cdot & \cdot & \cdot \\ 0 & \cdot & e^2 \end{bmatrix}$$

so that $\|E_1\|_F = e^2$, and $\|E_2\|_F \leqq 2s^2e^2/[gap(\sigma_{\min}(D)-|s|)] \leqq \eta s^2$; E_2 is in general nonsymmetric. By the Bauer–Fike theorem [8] each eigenvalues of $DD^T+E_1+E_2$ is within

$$\|E_1\|_F + \|E_2\|_F \leqq e^2 + \eta s^2 \leqq .5\eta(\sigma^2_{\min}(D)-s^2)+\eta s^2 \leqq \sigma^2_{\min}(D)$$

of an eigenvalue of DD^T. This completes the proof. □

This theorem justifies the following.

Convergence Criterion 2. Let $B=[\begin{smallmatrix} D & \tilde{e} \\ 0 & s \end{smallmatrix}]$ (or $B=[\begin{smallmatrix} s & \tilde{e}^T \\ 0 & D \end{smallmatrix}]$) where $\tilde{e}=[0,\cdots,0,e]^T$ (or $\tilde{e}=[e,0,\cdots,0]^T$). Let $\eta<1$ be the desired relative accuracy of the computed singular values. Then if $gap=\sigma_{\min}(D)-|s|>0$ and $|e|^2 \leqq .5\cdot\eta\cdot gap\cdot(\sigma_{\min}(D)+|s|) = .5\cdot\eta\cdot(\sigma^2_{\min}(D)-|s|^2)$, set e to zero. From (2.5), we may approximate $\sigma_{\min}(D)$ with the lower bound $\min_{j<n}\mu_j/(n-1)^{1/2}$ (or $\min_{j>1}\lambda_j/(n-1)^{1/2}$).

The following example shows that Convergence Criterion 2 may sometimes set e to zero before Convergence Criterion 1. Consider $B=[\begin{smallmatrix} 1 & e \\ 0 & .5 \end{smallmatrix}]$. Convergence Criterion 1 demands that $|e| \leqq \eta$ to set it to zero, whereas Convergence Criterion 2 demands only that $|e| \leqq (3\eta/8)^{1/2}$, which may be much larger.

This same example also shows why we do not want to use Convergence Criterion 2 when computing singular vectors. The right singular vectors of B and B' differ by $O(|e|)$, not $O(|e|^2)$.

In practice, we may estimate $\sigma_{\min}(D)$ using (2.4) and (2.5), and indeed we need only run the recurrences once to apply both Convergence Criteria 1 and 2.

3. QR iteration with a zero shift.

The standard algorithm for finding singular values of a bidiagonal matrix B is the QR algorithm applied implicitly to B^TB [7]. The algorithm computes a sequence B_i of bidiagonal matrices starting from $B_0=B$ as follows. From B_i the algorithm computes a shift σ^2, which is usually taken to be the smallest eigenvalue of the bottom 2-by-2 block of $B_iB_i^T$. Then the algorithm does an implicit QR factorization of the shifted matrix $B_i^TB_i-\sigma^2 I = \mathrm{QR}$, where Q is orthogonal and R upper triangular, from which it computes a bidiagonal B_{i+1} such that $B^T_{i+1}B_{i+1} = RQ+\sigma^2 I$. As i increases, B_i converges to a diagonal matrix with the singular values on the diagonal.

The roundoff errors in this algorithm are generally on the order of $\varepsilon\|B\|$, where ε is the precision of the floating point arithmetic used. From Theorem 3 of the last section, this means we would expect absolute errors in the computed singular values of the same order. In particular, tiny singular values of B could be changed completely.

In this section we present a variation of this standard algorithm that computes all the singular values of a bidiagonal matrix, even the tiniest ones, with guaranteed high relative accuracy. We will call this the "implicit zero-shift QR" algorithm, since it corresponds to the above algorithm when $\sigma = 0$. However, it is organized in such a way as to guarantee that each entry of B_{i+1} is computed from B_i to nearly full machine precision. Then Corollary 2 of the last section implies that the singular values of B_i and B_{i+1} all agree to high relative accuracy. When B_{i+1} has finally converged to a diagonal matrix, these diagonal entries must therefore also be accurate singular values for the initial $B = B_0$. Exactly how to detect this convergence is an interesting issue and discussed in the next section.

The rest of this section is organized as follows. First we review the standard algorithm for singular values of a bidiagonal matrix. Then we show how it simplifies when the shift is zero. Next we discuss an error analysis of the resulting implicit zero-shift QR algorithm which shows that it computes each entry of B_{i+1} with high relative accuracy (the details of the error analysis are in § 8). Finally, we discuss the asymptotic convergence rate.

The final algorithm is a hybrid of the standard QR and implicit zero-shift QR. Standard QR is used when the condition number of B (the ratio of the largest to smallest singular values) is modest. In this case the roundoff errors are guaranteed to make acceptably small perturbations in the smallest singular values of B. If the condition number is large, we use implicit zero-shift QR instead. The hybrid algorithm will be discussed more fully in the next section.

In order to summarize the standard QR algorithm, we need some notation. Let $J(i, j, \theta)$ denote the Given's rotation in entries i and j by angle θ. In other words, $J(i, j, \theta)$ is an n-by-n identity matrix except for rows and columns i and j whose intersections consist of the following 2-by-2 rotation matrix:

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}.$$

Given the vector x, choosing θ so that $x_j/x_i = \tan\theta$ means that the ith and jth entries of $J(i, j, \theta)x$ will contain $\pm\sqrt{(x_i^2 + y_i^2)}$ and 0, respectively.

We will illustrate the algorithm on a 4-by-4 example, where we use x and $+$ to indicate nonzero entries and 0 and blank to indicate zero entries. Initially B_i is in the form

$$B_i = \begin{bmatrix} x & x & & \\ & x & x & \\ & & x & x \\ & & & x \end{bmatrix}.$$

We begin by postmultiplying B_i by $J_1 \equiv J(1, 2, \theta_1)$, where θ_1 will be discussed in a moment. This introduces a nonzero entry in the $(2, 1)$ position:

$$B_i J_1 = \begin{bmatrix} x & x & & \\ + & x & x & \\ & & x & x \\ & & & x \end{bmatrix}. \tag{3.1}$$

B_iJ_1 will now be pre and postmultiplied by a sequence of Given's rotations whose purpose is to "chase the bulge" indicated by "+" off the end of the matrix. Choose θ_2 so that $J_2 \equiv J(1, 2, \theta_2)$ introduces a zero in the (2, 1) entry of $J_2B_iJ_1$:

$$J_2B_iJ_1 = \begin{bmatrix} x & x & + & \\ 0 & x & x & \\ & & x & x \\ & & & x \end{bmatrix}. \tag{3.2}$$

Next choose θ_3 in $J_3 \equiv J(2, 3, \theta_3)$, θ_4 in $J_4 \equiv J(2, 3, \theta_4)$, θ_5 in $J_5 \equiv J(3, 4, \theta_5)$, and θ_6 in $J_6 \equiv J(3, 4, \theta_6)$, to give the following sequence of transformations:

$$J_2B_iJ_1J_3 = \begin{bmatrix} x & x & 0 & \\ & x & x & \\ & + & x & x \\ & & & x \end{bmatrix}, \qquad J_4J_2B_iJ_1J_3 = \begin{bmatrix} x & x & & \\ & x & x & + \\ & 0 & x & x \\ & & & x \end{bmatrix},$$

$$J_4J_2B_iJ_3J_5 = \begin{bmatrix} x & x & & \\ & x & x & 0 \\ & & x & x \\ & & + & x \end{bmatrix}, \qquad B_{i+1} \equiv J_6J_4J_2B_iJ_1J_3J_5 = \begin{bmatrix} x & x & & \\ & x & x & \\ & & x & x \\ & & 0 & x \end{bmatrix}.$$

The usual error analysis of Given's rotations [18, p. 131-139] shows that the computed B_{i+1} is the exact transformation of a matrix $B_i + E$ where $\|E\|$ is on the order of $p(n) \in \|B_i\|$, $p(n)$ a modest function of n.

To choose θ_1 we compute a shift σ^2 that is generally the smallest eigenvalue of the bottom right 2-by-2 submatrix of $B_iB_i^T$. θ_1 is then chosen so that J_1 introduces a zero into the (2, 1) entry of $J_1^T(B_i^TB_i - \sigma^2 I)$. It is easy to see that this means that

$$\tan \theta_1 = \frac{(B_i^TB_i)_{12}}{\sigma^2 - (B_i^TB_i)_{11}}. \tag{3.3}$$

This choice of shift, called Wilkinson's shift, guarantees at least linear convergence and generally yields asymptotic cubic convergence of the off-diagonal entries of B_i to zero [15, p. 151]. This is assuming arithmetic is done exactly.

Now let us take $\sigma = 0$. Let us also drop the subscript i on B_i for simplicity of notation. From (3.3) we see that $\tan \theta_1 = -b_{12}/b_{11}$ so that the result of the first rotation (for a 4-by-4 matrix) is

$$B^{(1)} \equiv BJ_1 = \begin{bmatrix} b_{11}^{(1)} & 0 & & \\ b_{21}^{(1)} & b_{22}^{(1)} & b_{23} & \\ & & b_{33} & b_{34} \\ & & & b_{44} \end{bmatrix}.$$

We let the superscript on the matrix and its entries indicate that J_1 has been applied. Comparing to (3.1) we see that the (1, 2) entry is zero instead of nonzero. This zero will propagate through the rest of the algorithm and is the key to its effectiveness. After the rotation by J_2 we have

$$B^{(2)} \equiv J_2BJ_1 = \begin{bmatrix} b_{11}^{(2)} & b_{12}^{(2)} & b_{13}^{(2)} & \\ 0 & b_{22}^{(2)} & b_{23}^{(2)} & \\ & & b_{33} & b_{34} \\ & & & b_{44} \end{bmatrix}.$$

Note that

$$\begin{bmatrix} b_{12}^{(2)} & b_{13}^{(2)} \\ b_{22}^{(2)} & b_{23}^{(2)} \end{bmatrix} = \begin{bmatrix} \sin\theta_2 b_{22}^{(1)} & \sin\theta_2 b_{23} \\ \cos\theta_2 b_{22}^{(1)} & \cos\theta_2 b_{23} \end{bmatrix},$$

i.e., it is a rank-one matrix. Therefore, postmultiplication by J_3 to zero out the $(1, 3)$ entry will also zero out the $(2, 3)$ entry:

$$B^{(3)} \equiv J_2 B J_1 J_3 = \begin{bmatrix} b_{11}^{(2)} & b_{12}^{(3)} & 0 & \\ 0 & b_{22}^{(3)} & 0 & \\ & b_{32}^{(3)} & b_{33}^{(3)} & b_{34} \\ & & & b_{44} \end{bmatrix}.$$

Comparing to (3.2) we see that there is an extra zero on the superdiagonal. Rotation by J_4 just repeats the situation: the submatrix of $J_4 J_2 B J_1 J_3$ consisting of rows 2 and 3 and columns 3 and 4 is rank one, and rotation by J_5 zeros out the $(3, 4)$ entry as well as the $(2, 4)$ entry. This regime repeats itself for the length of the matrix.

The following algorithm incorporates this observation. It uses a subroutine *ROT*(f, g, cs, sn, r) which takes f and g as inputs and returns r, $cs = \cos\theta$ and $sn = \sin\theta$ such that

$$\begin{bmatrix} cs & sn \\ -sn & cs \end{bmatrix} \cdot \begin{bmatrix} f \\ g \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}. \tag{3.4}$$

ROT(f, g, cs, sn, r): takes f and g as input and returns cs, sn, and r, satisfying (3.4).

```
if ($f = 0$) then
    $cs = 0$; $sn = 1$: $r = g$
elseif ($|f| > |g|$) then
    $t = g/f$; $tt = \sqrt{1+t^2}$
    $cs = 1/tt$; $sn = t*cs$; $r = f*tt$
else
    $t = f/g$; $tt = \sqrt{1+t^2}$
    $sn = 1/tt$; $cs = t*sn$; $r = g*tt$
endif
```

Barring underflow and overflow (which can only occur if the true value of r itself would overflow), *ROT* computes cs, sn, and r to nearly full machine accuracy (see § 8 below for details). It also uses fewer operations than the analogous routine "rotg" in LINPACK [5].

IMPLICIT ZERO-SHIFT QR ALGORITHM. Let B be an n-by-n bidiagonal matrix with diagonal entries $s_1, \cdots, s_n$ and superdiagonal entries $e_1, \cdots, e_{n-1}$. The following algorithm replaces s_i and e_i by new values corresponding to one step of the QR iteration with zero shift:

```
$oldcs = 1$
$f = s_1$
$g = e_1$
for $i = 1, n-1$
    call $ROT(f, g, cs, sn, r)$
    if ($i \neq 1$) $e_{i-1} = oldsn*r$
    $f = oldcs*r$
    $g = s_{i+1}*sn$
    $h = s_{i+1}*cs$
```

```
        call $ROT(f, g, cs, sn, r)$
        $s_i = r$
        $f = h$
        $g = e_{i+1}$
        $oldcs = cs$
        $oldsn = sn$
    endfor
    $e_{n-1} = h*sn$
    $s_n = h*cs$
```

It is straightforward to verify that this algorithm "chases the bulge" in the manner described above. It is remarkable that outside the two calls to *ROT*, there are only four multiplications in the inner loop. This is to be contrasted with the usual QR algorithm, which in addition to two calls to *ROT* has twelve multiplications and four additions. Thus the inner loop is much more efficient than the standard algorithm. Note also that it is parallelizable, because $n/2$ rotations can be done at once. Since data need only be passed serially along the diagonal, it can also be implemented in a systolic array. However, the algorithms in § 6 seem much better suited to parallel processing.

This algorithm may be expressed in the following terser but equivalent form:

```
    $oldcs = 1$
    $cs = 1$
    for $i = 1, n-1$
        call $ROT(s_i*cs, e_i, cs, sn, r)$
        if $(i \neq 1)$ $e_{i-1} = oldsn*r$
        call $ROT(oldcs*r, s_{i+1}*sn, oldcs, oldsn, s_i)$
    endfor
    $h = s_n*cs$
    $e_{n-1} = h*oldsn$
    $s_n = h*oldcs$
```

The initial form will be more convenient for the error analysis in § 8.

This algorithm is also much more accurate than the standard algorithm. The source of the extra accuracy is the absence of possible cancellation, which means all roundoff errors appear multiplicatively (there is an addition in *ROT*, but it is harmless). Our model of arithmetic is the usual one:

$$fl(x \circ y) = (x \circ y) \cdot (1 + e), \tag{3.5}$$

where $\circ$ is one of $+, -, *$ and $/$, $fl(x \circ y)$ is the floating point result of the operation $\circ$, and $|e| \leqq \varepsilon$, where ε is the machine precision. This would appear to eliminate machines like the Cray and Cyber from consideration, since those machines do not conform to (3.5) for addition and subtraction when cancellation is involved, but since we only need to use (3.5) for multiplication (as well as square root, division, and addition of positive quantities in *ROT*), this analysis covers those machines as well. We also assume overflow and underflow do not occur (we return to these issues in § 8).

We present two theorems about the accumulation of error in the algorithm. The proofs are given in § 8. The first theorem develops a bound for the relative error in the computed s_i and e_i of the form $cn\varepsilon$ (c is a modest constant) and uses it with Theorem 2 of § 2 to show that the relative difference between the singular values of the bidiagonal matrix B and the output matrix B' of the implicit zero-shift QR algorithm is $cn^2\varepsilon/(1-cn^2\varepsilon)$. In other words, the relative error in the computed singular values can only grow with the square of the dimension.

THEOREM 6. *Let B be an n-by-n bidiagonal matrix and B' the matrix obtained by running the implicit zero-shift* QR *algorithm on B. Let the singular values of B be $\sigma_1 \geqq \cdots \geqq \sigma_n$, and the singular values of B' be $\sigma_1' \geqq \cdots \geqq \sigma_n'$. Then if*

$$\omega \equiv 69 n^2 \varepsilon < 1, \tag{3.6}$$

the relative differences between the singular values of B and the singular values of B' are bounded as follows:

$$|\sigma_i - \sigma_i'| \leqq \frac{\omega}{1-\omega}\,\sigma_i.$$

Let B_k be the matrix obtained after k repetitions of the implicit zero-shift QR *algorithm, and let $\sigma_{k1} \geqq \cdots \geqq \sigma_{kn}$ be its singular values. Then if condition* (3.6) *holds we have*

$$|\sigma_i - \sigma_{ki}| \leqq \left(\frac{1}{(1-\omega)^k} - 1\right) \cdot \sigma_i \approx 69 k n^2 \varepsilon \cdot \sigma_i,$$

where the approximation to the last upper bound holds if $k\omega \ll 1$.

This result is actually rather pessimistic, as our second result shows: when we approach convergence in the sense that all rotations are through angles bounded away from $\pi/2$, errors do not accumulate at all and the error in the computed e_i and s_i is bounded by $c' \cdot \varepsilon$, c' another modest constant. With Theorem 2 this yields an error bound on the computed singular values of the form $c'n\varepsilon/(1-c'n\varepsilon)$.

THEOREM 7. *Let B be an n-by-n bidiagonal matrix and B' the matrix obtained by running the implicit zero-shift* QR *algorithm on B. Assume that all the rotation angles θ during the course of the algorithm satisfy* $\sin^2\theta \leqq \tau < 1$. *Let the singular values of B be $\sigma_1 \geqq \cdots \geqq \sigma_n$, and the singular values of B' be $\sigma_1' \geqq \cdots \geqq \sigma_n'$. Then if*

$$\omega \equiv \frac{88 n \varepsilon}{(1-\tau)^2} < 1, \tag{3.7}$$

the relative differences between the singular values of B and the singular values of B' are bounded as follows:

$$|\sigma_i - \sigma_i'| \leqq \frac{\omega}{1-\omega}\,\sigma_i.$$

Let B_k be the matrix obtained after k repetitions of the implicit zero-shift QR *algorithm, where we assume all rotation angles θ satisfy* $\sin^2\theta \leqq \tau < 1$. *Let $\sigma_{k1} \geqq \cdots \geqq \sigma_{kn}$ be the singular values of B_k. Then if condition* (3.7) *holds we have*

$$|\sigma_i - \sigma_{ki}| \leqq \left(\frac{1}{(1-\omega)^k} - 1\right) \cdot \sigma_i \approx \frac{88 k n \varepsilon}{(1-\tau)^2} \cdot \sigma_i,$$

where the approximation to the last upper bound holds if $k\omega \ll 1$.

Note that τ can easily be monitored by the algorithm as it proceeds.

The standard algorithm does not always achieve this accuracy for three reasons. First, the convergence criteria in the standard algorithm can change small singular values completely (this is discussed in detail in the next section). Second, rounding errors committed while "chasing the bulge" with a large shift can obscure small matrix entries and small singular values. Third, roundoff errors when the shift is zero result in nonzero entries appearing and propagating in those off-diagonal entries of intermediate results that should be zero, and that are kept zero by the new algorithm. This third effect seems mild, however, and as a result the standard algorithm sometimes computes small singular values with higher relative accuracy than the usual bound $p(n)\varepsilon\|A\|$ would lead us to expect (see, for example, the numerical examples of Class 1 in § 7).

The pattern of zeros above the diagonal during the QR sweep also appears when applying QR to a symmetric tridiagonal matrix. This pattern can be exploited to give fast, square root free versions of the algorithm (see [15, p. 164] for a discussion). Unfortunately, this does not yield forward stability and high accuracy as it does for the bidiagonal case.

Finally, we discuss the asymptotic convergence rate of the algorithm. It is well known that unshifted QR on a symmetric matrix is essentially the same as inverse iteration [15, p. 144]. Therefore we can conclude that the last off-diagonal element e_{n-1} should converge to zero linearly with constant factor $\sigma_{n-1}^2/\sigma_n^2$. If there is a cluster of m small singular values isolated from the remaining ones, e_{n-m} will converge to zero linearly with constant factor $\sigma_{n-m+1}^2/\sigma_{n-m}^2$.

4. Convergence criteria. In this section we discuss convergence criteria for the new algorithm, and describe the practical version of the algorithm, which is a hybrid of the usual shifted QR and the implicit zero-shift QR. After showing that the LINPACK convergence criteria [5] are unsatisfactory, we restate the convergence criteria of § 2. The same analysis leading to the convergence criteria will lead to a criterion for switching from zero-shift QR to shifted QR without damaging any tiny singular values. The switching criterion depends on a user specifiable tolerance *tol*, which is the desired relative accuracy in the singular values (*tol* should be less than 1 and greater than the machine precision ε). The resulting hybrid algorithm will therefore run about as fast as the standard algorithm on matrices without any tiny singular values. We will also discuss convergence criteria in the case where we are only interested in absolute precision in the singular values. Finally, we discuss the impact of underflow on the convergence criteria.

We begin by discussing the convergence criteria used in the current version of the algorithm [5], and explain why they are unsuitable for our algorithm. The code in LINPACK has two tests for setting entries of the bidiagonal matrix B to zero. Recall that $s_1, \cdots, s_n$ are the diagonal entries of B and $e_1, \cdots, e_{n-1}$ are the off-diagonal entries. The first test is

$$\text{if } (|e_i|+|e_{i-1}|+|s_i|=|e_i|+|e_{i-1}|), \quad \text{set } s_i \text{ to } 0. \tag{4.1}$$

This rather enigmatic looking test works as follows. If $|s_i|<.5\cdot\varepsilon\cdot(|e_i|+|e_{i-1}|)$, the test will be satisfied and s_i set to zero. In other words, (4.1) is a way of asking whether one number is less than roundoff error in another number without needing to know the machine precision ε explicitly. The other convergence test is

$$\text{if } (|s_i|+|s_{i-1}|+|e_{i-1}|=|s_i|+|s_{i-1}|), \quad \text{set } e_{i-1} \text{ to } 0. \tag{4.2}$$

Both tests compare an entry x of B with its two nearest neighbors on the other diagonal, and set x to zero if it is negligible compared to those neighbors. One justification for these tests is that roundoff error during the rotations could make the matrix indistinguishable from one with a zero in x's position. Also, they clearly introduce errors no worse than $p(n)\varepsilon\|A\|$. (Both these tests may be unnecessarily slow for these purposes on machines where the quantities $|e_i|+|e_{i-1}|+|s_i|$, $|e_i|+|e_{i-1}|$, $|s_i|+|s_{i-1}|+|e_{i-1}|$ and $|s_i|+|s_{i-1}|$ are computed and compared in extended precision registers, where the effective ε is much tinier than in working precision.)

Both tests are unsatisfactory for our algorithm. Test (4.1) introduces a zero singular value where there was none before, so it is clearly unsatisfactory. The following example shows why (4.2) is also unsatisfactory. Suppose η is sufficiently small that in floating point arithmetic $1+\eta=1$. Consider the matrix

$$B(x)=\begin{bmatrix} \eta^2 & 1 & & \\ & 1 & x & \\ & & 1 & 1 \\ & & & \eta^2 \end{bmatrix}.$$

When $x=\eta$ it is easy to verify that the smallest singular value of $B(\eta)$ is about η^3. Test (4.2) would set x to 0, but $B(0)$ has a smallest singular value of about $\eta^2/\sqrt{2}$, which is utterly different.

Our convergence criteria must guarantee that by setting some e_i to 0 (clearly no nonzero s_i can ever be set to zero), no singular value is perturbed too much. Let $\underline{\sigma}$ denote a reliable estimate or underestimate of the smallest singular value. Such a $\underline{\sigma}$ is provided by the recurrences for $\|B^{-1}\|_\infty^{-1}$ and $\|B^{-1}\|_1^{-1}$ in (2.4). Then the simplest acceptable convergence criterion would only set e_i to zero if it were less than $tol*\underline{\sigma}$. However, this method is overly conservative, and generally waits much too long to set e_i to 0. Much better estimates are given in § 2 and justified by Theorems 4-5. We repeat them here.

Convergence Criterion 1a. Let μ_j be computed by the following recurrence ((2.4) from § 2):

```
(4.3)        $\mu_1 = |s_1|$
             for $j=1$ to $n-1$ do
                 $\mu_{j+1} = |s_{j+1}| \cdot (\mu_j/(\mu_j+|e_j|))$
```

If $|e_j/\mu_j| \leqq tol$, set e_j to 0.

Convergence Criterion 1b. Let λ_j be computed by the following recurrence ((2.4) from § 2):

```
(4.4)        $\lambda_n = |s_n|$
             for $j=n-1$ to 1 step $-1$ do
                 $\lambda_j = |s_j| \cdot (\lambda_{j+1}/(\lambda_{j+1}+|e_j|))$
```

If $|e_j/\lambda_{j+1}| \leqq tol$, set e_j to 0.

Convergence Criterion 2a. Let μ_j be computed from (4.3). If singular vectors are not desired, and $e_{n-1}^2 \leqq .5 \cdot tol \cdot [(\min_{j<n} \mu_j/(n-1)^{1/2})^2 - |s_n|^2]$, set e_{n-1} to zero.

Convergence Criterion 2b. Let λ_j be computed from (4.4). If singular vectors are not desired, and $e_1^2 \leqq .5 \cdot tol \cdot [(\min_{j>1} \lambda_j/(n-1)^{1/2})^2 - |s_1|^2)$, set e_1 to zero.

We have divided the criteria of § 2 into separate parts, because we will apply them in separate situations; see § 5.2.

These criteria are more expensive than the standard LINPACK criteria, but avoid situations such as setting x to 0 in $B(x)$ in the last paragraph, and recognizes that setting x to zero in

$$\begin{bmatrix} 1 & x \\ & D \end{bmatrix}$$

is harmless if $|x| \leqq tol$, independent of D.

Now we consider how to decide whether to use implicit zero-shift QR or standard shifted QR. In order to estimate the rounding errors that would occur during shifted QR, we need an estimate of $\|B\|$. We will use $\bar{\sigma} \equiv \max_i (|s_i|, |e_i|)$, which is easily seen to underestimate $\|B\|$ by no more than a factor of 2. In terms of $\underline{\sigma}$, $\bar{\sigma}$, and *tol*, our decision algorithm is

```
if ($fudge * tol * (\underline{\sigma}/\bar{\sigma}) \leqq \varepsilon$)
    use the implicit zero-shift QR
else
    use shifted QR
endif
```

The test asks whether the rounding errors $\varepsilon \cdot \bar{\sigma}$ that shifted QR could introduce are greater than the largest tolerable perturbation $tol \cdot \underline{\sigma}$. The factor $fudge \geqq 1$ is a fudge factor that makes zero-shifting less likely on tight clusters of singular values; we currently use $fudge = \min(n, m)$ if the original matrix was n by m.

In practice there is one further test for using the implicit zero-shift QR. If the above test chooses shifted QR, we must still compute the shift σ^2, which is the smallest eigenvalue of the bottom 2-by-2 matrix of BB^T. From (3.3), we see that the tangent of the first rotation angle is given by

$$\frac{s_1 * e_1}{\sigma^2 - s_1^2} = \frac{e_1}{s_1}\left(\frac{\sigma^2}{s_1^2} - 1\right)^{-1}$$

so if $\sigma^2/s_1^2 - 1$ rounds to -1, the first rotation is the same as in implicit zero-shift QR and we might as well use it, since it is faster and more accurate.

The choice of *tol* may be made by the user, or chosen automatically by the program. If *tol* is chosen close to 1, we almost always pick shifted QR, which still computes the singular values with good absolute accuracy, so only the smallest singular values will be inaccurate. If we choose *tol* near ε, we will almost always use implicit zero-shift QR unless all the singular values are very close together, and therefore sacrifice the cubic convergence of shifted QR. See § 7 for descriptions of numerical experiments on the effect of varying ε. Choosing *tol* near 1 is useful for quickly obtaining estimates of singular values for rank determination. Note that as singular values converge and are deflated off, $\underline{\sigma}$ may be reestimated so that if *tol* is not too small, by the time all the small singular values have converged, the algorithm is doing shifted QR.

Note also that if we are only interested in computing the smallest singular value or values, $\underline{\sigma}$ provides a test for stopping the iteration early. If one or several small singular values have been deflated out, and the $\underline{\sigma}$ for the remaining matrix exceeds them sufficiently, we are guaranteed to have found the smallest singular value. A similar idea is expressed in [17].

Finally, we consider computing the singular values to guaranteed absolute accuracy instead of guaranteed relative accuracy. As stated in the Introduction, this is what standard shifted QR guarantees. However, the convergence criteria (4.1) and (4.2) in the current standard implementation are much more stringent than necessary to meet this goal. Instead of comparing $|e_i|$ or $|s_i|$ to its neighbors to see if it is negligible, it is only necessary to compare to $\bar{\sigma} \approx \|B\|$. In other words substituting

(4.5) $$\text{if } (|s_i| \leqq tol * \bar{\sigma}) \quad \text{set } s_i \text{ to } 0$$

for (4.1) and

(4.6) $$\text{if } (|e_i| \leqq tol * \bar{\sigma}) \quad \text{set } e_i \text{ to } 0$$

for (4.2) will also guarantee absolute accuracy but possibly speed convergence considerably. In practice, our code uses the input parameter *tol* to choose between absolute and relative accuracy: if *tol* is positive, it indicates that relative accuracy *tol* is desired, and if *tol* is negative, it indicates that absolute accuracy $|tol| \cdot \bar{\sigma}$ is desired.

Underflow must also be accounted for in the convergence criteria to ensure convergence. For it may happen that the quantity to be subtracted from e_{n-1} in the course of driving it to zero may underflow, so that e_{n-1} never decreases. On machines with IEEE arithmetic, this may occur if all entries of B are denormalized. To prevent this, we make sure the convergence threshold to which we compare $|e_j|$ is at least $maxit^*\lambda$, where *maxit* is the maximum number of QR inner loops the code will perform, and λ is the underflow threshold (the smallest positive normalized number). If the matrix has singular values near λ or smaller, this technique could destroy their accuracy; in this case the matrix should be scaled up by multiplying it by $maxit/\varepsilon$ before applying the algorithm, and multiplying the computed singular values by $\varepsilon/maxit$ afterwards.

5. Implementation details. In this section we discuss a number of details of the implementation of the code: Chasing the bulge up or down; applying the convergence criteria; SVD of 2-by-2 triangular matrices and robust shift calculation; deflation when $s_i = 0$.

Finally, we present high-level code for the entire algorithm.

5.1. Chasing the bulge up or down. A bidiagonal matrix may be graded in many ways, but most commonly it will be large at one end and small at the other. The implicit zero-shift QR algorithm tries to converge the singular values in order from smallest to largest. If the matrix is graded from large at the upper left to small at the lower right, and the "bulge" is chased from upper left to lower right as in § 3, then convergence will be fast because the singular values are "ordered" correctly, i.e., the diagonal matrix entries are fairly close to their final values. If, however, the matrix is graded the opposite way (from small at the left to large at the right) then the algorithm will have to invert the order of the matrix entries as it converges. This may require many more QR steps. To avoid this, the implementation tests for the direction of grading (simply comparing $|s_1|$ and $|s_n|$), and chases the bulge in the direction from large to small. If a matrix breaks up into diagonal blocks that are graded in different ways, the bulge is chased in the appropriate direction on each block. The algorithm in [17] does this as well.

In order to avoid the possibility that the code might frequently change bulge chasing directions, and so converge very slowly, we only choose the direction of bulge chasing when beginning work on a submatrix disjoint from the previous one. Whether this is the optimal strategy is a question of future research.

This means the singular values may be quite disordered in the final converged matrix, and so must be sorted at the end (along with the singular vectors if desired). The LINPACK SVD uses bubble sort at the end, which could require $O(n^2)$ swaps of singular vectors. Since the LINPACK SVD always chases the bulge down, the singular values tend to converge in nearly sorted order, so bubble sort is relatively efficient. The new algorithm, in which the singular values could converge in any order, uses insertion sort instead, which does at most $2n$ moves of singular vectors.

5.2. Applying the convergence criteria. In § 4 we presented four convergence criteria. Since applying the convergence criteria costs approximately as many floating point operations ($O(n)$) as performing a QR sweep, it is important to test criteria only when they are likely to be satisfied. Our decision is based on the following empirical observation: When chasing the bulge down (up), the bottommost (topmost) entry $s_n(s_1)$ often tends to converge to the smallest singular value, with $e_{n-1}(e_1)$ tending to zero fastest of all off-diagonal entries. Therefore, when chasing the bulge down, we expect Convergence Criteria 1a and 2a to be successful, and possibly 1b but only for the bottommost entry e_{n-1}. Convergence Criteria 2b and 1b for the other off-diagonal entries are not as likely to succeed. Conversely, when chasing the bulge up, we only apply Convergence Criteria 1b, 2b, and 1a for e_1. One advantage of this scheme is that testing 2a (for e_{n-1}, and if the test succeeds, for e_{n-2} too) costs only a few more operations after testing 1a, since they share the same recurrence from (4.3). Similarly, 2b (for e_1, and if the test succeeds, for e_2 too) is very cheap after applying 1b.

5.3. SVD of 2-by-2 triangular matrices and robust shift calculation. The need for the singular value decomposition of 2-by-2 triangular matrices, or at least the smallest singular value of such a matrix, arises in two places in the code. The first time is when calculating the shift. As stated in § 3, the standard choice of shift, called Wilkinson's shift, is the smallest eigenvalue of the bottom 2-by-2 block of BB^T. It is easy to see that this is the square of the smallest singular value of the bottom 2-by-2 block of B. The second need for the SVD of a 2-by-2 triangular matrix arises when the code has isolated a 2-by-2 block on the diagonal of B. Even though this appears to be an easy case for the algorithm in § 4, it turns out that roundoff can prevent convergence when the singular values are close. This is the case in

$$B=\begin{bmatrix} a & b \\ 0 & c \end{bmatrix}$$

when $|a|$ and $|c|$ are close and b is much smaller, just larger than $\varepsilon \cdot |a|$. It happens that on machines with sloppy arithmetic, roundoff can cause b to be no smaller after one step of QR than before, so that the algorithm never converges. It is also difficult in this situation to compute the singular vectors accurately, just as eigenvectors corresponding to multiple eigenvalues are difficult to compute.

To get around these difficulties, we have written a subroutine that takes the entries a, b, and c of B and returns the two singular values as well as the left and right singular vectors. Barring overflow and underflow, the returned values are accurate to nearly full machine precision, even for nearly coincident singular values. The algorithm is comparable in speed to a straightforward implementation that does not attain similar accuracy. This property is based on the fact that the algorithm uses formulas for the answer that contain only

products, quotients, and square roots,
sums of terms of like sign,
differences of computed quantities only when cancellation is impossible, and
the difference $|a|-|c|$, which, if cancellation occurs, is exact (except possibly on a Cray or Cyber).

It is straightforward to use these properties to show that the final result is correct to nearly full precision.

The code is also robust in the face of over/underflow. Overflow is avoided where possible by using formulas in terms of ratios of matrix entries, and choosing the formulas so that the ratios are always bounded by 1 in magnitude. As a result of these precautions, overflow is impossible unless the exact largest singular value itself overflows (or is within a few units in the last place of overflowing). Underflow (of the conventional "store zero" variety) can damage the results only if the data and/or results are themselves close to the underflow threshold, specifically less than the underflow threshold divided by ε. Gradual underflow [2] makes the calculation of the singular values impervious to underflow (unless the final results themselves underflow) and the singular vectors much less susceptible to underflow problems.

5.4. Deflation when $s_i=0$. The standard SVD algorithm [5] has special code to handle the case when $s_i=0$. This code does a simplified sequence of rotations (similar to implicit zero-shift QR) to introduce a zero on the superdiagonal of the bidiagonal matrix (adjacent to the zero on the diagonal) and so break it into two smaller problems. It is easy to see that the implicit zero-shift QR algorithm does this deflation automatically, yielding one zero on the superdiagonal for each zero on the diagonal, but at the bottom (or top) of the matrix, rather than where the original zero occurred. This occurs after one pass of the algorithm, at which point both s_n and e_{n-1} will be zero if chasing the bulge down (s_1 and e_1 will be zero if chasing the bulge up) meaning that the zero singular value has been deflated exactly.

We can see this as follows. Assume we are chasing the bulge down. Whenever $s_{i+1}=0$, both g and h will be set to 0, causing the sn returned by the second call to *ROT* to be 0. At the end of the loop, both $f=h$ and $oldsn=sn$ will also be zero. In fact, it is easy to see that from now on both h and the f at the bottom of the loop will be zero: at the top of the next loop iteration, the zero value of f causes the first call of *ROT* to compute $cs=0$; this causes $h=s_{i+1} * cs$ to be zero and the pattern repeats. Also, when $oldsn=0$ (which happens when $s_{i+1}=0$), e_{i-1} is set to zero on the next iteration, i.e., $s_{i+1}=0$ implies e_i becomes zero. Finally, at the end of all the loop iterations, h is still zero implying both e_{n-1} and s_n are set to zero. Note that when f is zero, as it frequently is in this case, the first call to *ROT* need only set $cs=0$, $sn=1$, and $r=g$; this is what the first "if" branch in *ROT* does.

Finally, we present a high-level description of the entire algorithm. In the interest of brevity we omit the code for updating the singular vectors or for the absolute error convergence criterion.

ε = machine precision
λ = underflow threshold (smallest positive normalized number)
n = dimension of the matrix
tol = relative error tolerance (currently 100ε)
$maxit$ = maximum number of QR inner loops (currently $3n^2$)

Bidiagonal singular value decomposition.

```
Compute $\underline{\sigma} \leqq \sigma_{\min}(B)$ using (2.4)
$\bar{\sigma} = \max(|s_i|, |e_i|)$
$thresh = \max(tol \cdot \underline{\sigma}, maxit \cdot \lambda)$
/* any $e_i$ less than $thresh$ in magnitude may be set to zero */
Loop:
    /* Find bottommost nonscalar unreduced block diagonal submatrix of $B$ */
    let $\bar{i}$ be the smallest $i$ such that $|e_i|$ through $|e_{n-1}|$ are at most $thresh$, or $n$ if
    no such $i$ exists
    if $\bar{i}=1$, goto Done
    let $i'$ be the largest $i$ less than $\bar{i}$ such that $|e_i| \leqq thresh$, or 0 if no such $i$ exists
    $\underline{i} = i'+1$
    /* Apply algorithm to unreduced block diagonal submatrix from $\underline{i}$ to $\bar{i}$ */
    if $\bar{i} = \underline{i}+1$, then
        /* 2-by-2 submatrix, handle specially */
        compute SVD of 2-by-2 submatrix, setting $e_{\underline{i}}$ to 0
        goto Loop
    endif
    if submatrix from $\underline{i}$ to $\bar{i}$ disjoint from submatrix of last pass through Loop, then
        /* Choose bulge chasing direction */
        if $|s_{\underline{i}}| \geqq |s_{\bar{i}}|$, then
            $direction$ = "$down$"
        else
            $direction$ = "$up$"
        endif
    endif
    /* Apply convergence criteria */
    if $direction$ = "$down$", then
        Apply convergence criterion 1b to $e_{\bar{i}-1}$
        Apply convergence criterion 1a
        Apply convergence criterion 2a to $e_{\bar{i}-1}$ and possibly $e_{\bar{i}-2}$
    else
        Apply convergence criterion 1a to $e_{\underline{i}}$
        Apply convergence criterion 1b
        Apply convergence criterion 2b to $e_{\underline{i}}$ and possibly $e_{\underline{i}+1}$
    endif
    /* Compute shift */
    if $fudge*tol*\underline{\sigma}/\bar{\sigma} \leqq \varepsilon$, then
        /* Use zero shift because tiny singular values present */
        shift = 0
    else
        if $direction$ = "$down$", then
            $s = s_{\bar{i}}$
            $shift$ = smallest singular value of bottom 2-by-2 corner
        else
            $s = s_{\underline{i}}$
            $shift$ = smallest singular value of top 2-by-2 corner
        endif
        if $(shift/s)^2 \leqq eps$, then
            /* Use zero shift, since shift rounds to 0 */
            $shift = 0$
        endif
    endif
    /* Perform QR iteration */
    if $shift = 0$, then
        if $direction$ = "$down$", then
            do implicit zero-shift QR downward
            if $|e_{\bar{i}-1}| \leqq thresh$, set $e_{\bar{i}-1} = 0$
        else
            do implicit zero-shift QR upward
            if $|e_{\underline{i}}| \leqq thresh$, set $e_{\underline{i}} = 0$
        endif
    else
        if $direction$ = "$down$", then
            do standard shifted QR downward
            if $|e_{\bar{i}-1}| \leqq thresh$, set $e_{\bar{i}-1} = 0$
        else
            do standard shifted QR upward
            if $|e_{\underline{i}}| \leqq thresh$, set $e_{\underline{i}} = 0$
        endif
    endif
    goto Loop
    Done: sort singular values
```

6. Other methods for computing accurate singular values. In this section we discuss other methods for computing the singular values of a bidiagonal matrix B to high relative accuracy. These methods include bisection, Rayleigh quotient iteration, and iterative refinement. They are not competitive in speed with QR for computing all the singular values on a serial machine, but can efficiently verify whether or not a computed singular value is accurate. We have used them to verify the numerical results presented in § 7. However, they are extremely easy to parallelize and will probably be among the best parallel algorithms for this problem.

All the algorithms are based on bisection for the symmetric tridiagonal eigenproblem, which we discuss first. Bisection is in turn based on Sylvester's Law of Inertia, or equivalently, Sturm sequences [15, p. 52]. As explained in § 2, the number of negative d_i in recurrence (2.1) is the number of eigenvalues less than x, a quantity we will denote by $v(x)$. $v(x_2)-v(x_1)$ is therefore the number of eigenvalues in the interval $[x_1, x_2)$, so this method can easily verify whether an interval contains any eigenvalues. The identification of the singular value problem for the bidiagonal matrix B with the eigenvalue problem for a symmetric tridiagonal matrix with zero diagonal later in § 2 makes it clear that we can use the same method to count the number of singular values in any interval $[x_1, x_2)$.

What remains is an error analysis to show that the function $v(x)$ is accurate. This is provided in [12, p. 35]:

Let $v(x)$ be the computed number of eigenvalues less than x for a symmetric tridiagonal matrix A. Barring over/underflow, the computed value of $v(x)$ is the exact value of $v(x)$ for the perturbed matrix $A+\delta A$ where $|\delta A| \leqq 2\varepsilon|\text{offdiag}(A)| + \varepsilon x I$. Here, offdiag (A) refers to the off-diagonal part of A. If A has a zero diagonal, this bound may be improved to $|\delta A| \leqq 1.5 \cdot \varepsilon |A|$.

Therefore, by Theorem 2 or Corollary 2, if the computed value of $v(x)$ is k, there must be at least k singular values of B less than $x/(1-(3n-1.5)\varepsilon)$ and no more than k singular values less than $x \cdot (1-(6n-2)\varepsilon)/(1-(3n-1.5)\varepsilon)$; we assume $n\varepsilon \ll 1$. If the computed value of $v(x_2)-v(x_1)$ is j, there must be at least j singular values in the interval $[x_1 \cdot (1-(6n-2)\varepsilon)/(1-(3n-1.5)\varepsilon), x_2/(1-(3n-1.5)\varepsilon)]$.

There is one other important feature of the computed $v(x)$. In exact arithmetic, since $v(x)$ is the number of eigenvalues less than x, $v(x)$ must be a monotonic increasing function of x. It is by no means clear that the computed values of $v(x)$ should also be monotonic. This is significant because a failure in monotonicity could cause an algorithm to misestimate the number of eigenvalues in an interval, although a bisection routine that begins with an interval $[x_1, x_2)$ where $v(x_2)-v(x_1)$ is positive can always maintain an interval over which the computed value of v increases. It turns out, however, that as long as the arithmetic is monotonic, the computed value of $v(x)$ will be monotonic [12, p. 27]. By monotonic arithmetic we mean that if $a \circ b \geqq c \circ d$ in exact arithmetic, then $fl(a \circ b) \geqq fl(c \circ d)$ as well. This holds in any well-designed arithmetic, such as the IEEE Floating Point Standard 754 [10]. We have only shown that monotonicity holds if the recurrence is computed exactly as follows, with the order of evaluation respecting parentheses:

$$d_i = (a_i - (b_{i-1}^2/d_{i-1})) - x.$$

Now we briefly consider Rayleigh quotient iteration and iterative refinement. Both algorithms begin with a small interval containing a singular value, and refine it as does bisection. The major difference with bisection is in the zerofinder used to refine the intervals. As long as they are implemented in a componentwise backward stable way (i.e., they compute the correct result for a bidiagonal having only small relative perturbations in each entry), then Corollary 2 and Theorem 2 guarantee the relative accuracy of the computed singular values.

7. Numerical experiments. The numerical experiments we discuss here compare the algorithm of §§ 3-5 with the LINPACK SVD [5]. Both codes were run in double precision on a SUN 3/60 with an MC68881 floating point coprocessor, which implements IEEE standard 754 floating point arithmetic [10]; the machine precision $\varepsilon = 2^{-53} \approx 1.1 \cdot 10^{-16}$ and the range is approximately $10^{\pm 308}$. In order to guarantee reliable timings, each matrix tested was run sufficiently often that the total elapsed time was about 10 seconds. Singular vectors were computed by identical calls to *drot* [5] in both algorithms.

The codes were compared with respect to

accuracy,
total number of passes through the inner loop of QR iteration,
(half the number of Givens rotations performed)
elapsed time when computing singular values only,
elapsed time when computing both left and right singular vectors as well,
elapsed time including bidiagonalizing the input matrix, and
elapsed time excluding bidiagonalizing the input matrix.

Also, the dependence of the new algorithm on the parameter *tol* (see § 4) was investigated. At the end we comment on the implications of our results for the "perfect shift" strategy for computing singular vectors.

The LINPACK code was modified to explicitly use the machine precision ε in the stopping criteria rather than implicitly as in (4.1) and (4.2). Specifically,

$$\text{if } (|s_i| \leq \varepsilon * (|e_i| + |e_{i-1}|)), \quad \text{set } s_i \text{ to } 0 \tag{7.1}$$

was used in place of (4.1) and

$$\text{if } (|e_{i-1}| \leq \varepsilon * (|s_i| + |s_{i-1}|)), \quad \text{set } e_{i-1} \text{ to } 0 \tag{7.2}$$

was used in place of (4.2). Thus since both the new algorithm and modified LINPACK code use stopping criteria with ε appearing explicitly, there is no danger that the extended precision registers on the MC68881 would cause tests like (4.1) and (4.2) to be executed with a smaller effective ε than expected, which could slow convergence.

The LINPACK code also used a corrected shift calculation rather than the erroneous one in [5]. The version in [5] computes $f = (sl + sm)*(sl - sm) - shift$; this should be $f = (sl + sm)*(sl - sm) + shift$ instead (the corrected version is distributed by netlib [4]).

It turns out that the results depend strongly on the form of the bidiagonal matrix. For example, the standard SVD behaves entirely differently on matrices graded from top to bottom than on matrices graded in the opposite direction. Therefore, we present our results on twelve separate classes of bidiagonal matrices, since this seems to be the only fair way to compare results. The classes are as follows.

Class 1. These eight matrices are graded in the usual way from large at the upper left to small at the lower right. All matrices have a 1 in the upper corner, and each superdiagonal entry $B_{i,i+1}$ equals its neighbor B_{ii} on the diagonal. Four of the matrices are 10-by-10 and have a constant multiple between adjacent entries on the diagonal and superdiagonal: 10^{10}, 10^{5}, 10^{2}, and 10. The other four are 20-by-20 and are obtained from the first four by simply repeating each entry once, e.g., a diagonal containing 1, $10^{-10}, 10^{-20}, \cdots, 10^{-90}$ becomes $1, 1, 10^{-10}, 10^{-10}, 10^{-20}, 10^{-20}, \cdots, 10^{-90}, 10^{-90}$.

Class 2. This class is identical to Class 1 except the order of the entries on the diagonal and superdiagonal are reversed. Thus these matrices are graded from small at the upper left corner to large at the lower right.

Class 3. These eight 20-by-20 and 40-by-40 matrices are obtained by abutting those in Class 1 with their reversals in Class 2. Thus each matrix is small at the upper left, large in the middle, and small again at the lower right.

Class 4. These eight 20-by-20 and 40-by-40 matrices are obtained by abutting those in Class 2 with their reversals in Class 1. Thus each matrix is large at the upper left, small in the middle, and large again at the lower right.

Class 5. These eight matrices are obtained from Class 1 by reversing the order of the superdiagonals. Thus the diagonal is graded from large at the upper left to small at the lower right, and the superdiagonal is graded in the opposite direction.

Class 6. These eight matrices are obtained from Class 5 by reversing the order of both the diagonals and superdiagonals. Thus the diagonal is graded from small at the upper left to large at the lower right, and the superdiagonal is graded in the opposite direction.

Class 7. These sixteen matrices are all small on the diagonal and mostly large on the off-diagonal. Eight of them are 10-by-10 with 1's on the off-diagonal and a constant diagonal, equaling 10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10}, 10^{-12}, 10^{-14}, and 10^{-16}, respectively. The other eight 20-by-20 matrices are obtained by putting two copies of each of the first eight together, and "connecting" them by setting the middle off-diagonal entry $B_{10,11}$ to be 10^{-15} times the value of the diagonal entries.

Classes 8-11. The ten 20-by-20 matrices in each class are generated by letting each bidiagonal entry be a random number of the form $r \cdot 10^i$, where r is a random number uniformly distributed between $-.5$ and $.5$, and i is a random integer. In Class 8, i is uniformly distributed from 0 to -15. In Class 9, i is uniformly distributed from 0 to -10. In Class 10, i is uniformly distributed from 0 to -5. In Class 11, i is identically 0. Thus in Class 11 each matrix entry is simply uniformly distributed on $[-.5, .5]$.

Class 12. This one 41-by-41 matrix is graded as in Class 1, with the ratio of adjacent entries being $10^{-.1} \approx .79$. Each off-diagonal entry is identical to the diagonal entry below it. This very dense grading leads to different convergence properties than for the matrices in Class 1, which is why we put this example in a separate class.

Thus Classes 1-6 and 12 consist of graded matrices, Class 7 consists of matrices larger on the off-diagonal than the diagonal, and Classes 8-11 consist of random matrices with random exponents.

First we discuss the accuracy of computed singular values. With $tol = 100\varepsilon \approx 10^{-14}$ the new algorithm always converged in fewer than $maxit = 3n^2$ passes through the QR inner loop and computed all singular values to nearly full accuracy. Accuracy was determined using the method in § 6: If σ is a computed singular value, the number of singular values in the interval $[\sigma(1-n\varepsilon), \sigma(1+n\varepsilon))$ were counted. Overlapping intervals were joined into larger intervals. The number of computed singular values in each interval was then compared with the true number of singular values in each interval. This accuracy test was passed in all cases but one singular value out of 2041 singular values of all 105 matrices. In other words, 2040 singular values were computed with a relative error of about 10^{-14} or better; the exceptional singular value (in Class 11) had a relative error a little less than $3 \cdot 10^{-14}$.

The accuracy of the singular values computed by the LINPACK SVD were determined by comparison with the singular values from the new algorithm. This data is presented in Table 1. We called agreement to at least fourteen digits with the verified correct results of the new algorithm "all digits correct"; the notation "% all digits" in Table 1 means the percentage of such singular values. The notation "% $m-n$ digits" in Table 1 means the percentage of singular values computed with m to n correct digits. 0 digits means that the order of magnitude is still correct. -1 digits means correct to within a factor of 10. The column "% nonzero, no digits" gives the percentage of computed singular values that were nonzero and had incorrect orders of magnitude. The column "% zero, no digits" gives the percentage of computed singular values that were exactly zero, even though the matrix was nonsingular.

One striking feature about this table is the difference between Classes 1 and 2. The only difference between the matrices in Classes 1 and 2 is the order of the entries. When the entries are graded from large to small, the standard SVD gets all the singular values correct. Indeed, it was constructed to perform well on matrices graded in this fashion. When they are graded in the opposite way, only 42 percent are fully correct and another third have fewer than four digits correct. Three percent are computed as 0 even though all matrices tested were nonsingular. This happens because the standard SVD always "chases the bulge" from top to bottom. When the matrix is graded from large to small, this works well, but when it is graded in the opposite way as in Class 2, the algorithm must "reorder" all the matrix entries, and in doing so must combine tiny entries with large entries, thereby losing precision. The same thing happens for Class 4. The new algorithm avoids the need to reorder by always "chasing the bulge" from the large to the small end of the matrix. This is also done in the algorithm in [17] (see § 5 for details). The nonzero singular values that are not even order of magnitude correct are off by factors of 10^{-53} and 10^{-57} (Class 7) and 10^{-5} (Class 8). The last column indicates how often the computed singular values were exactly zero, when in fact none of the test matrices were singular.

We evaluated the computed singular vectors by computing the norm of the residual $BV-U\Sigma$, where B is the bidiagonal matrix, V contains the right singular vectors, U the left singular vectors, and Σ the singular values. The norm was the maximum absolute matrix entry. In all cases for both new and old SVD this measure never exceeded $1.1\cdot 10^{-14}\approx 100\varepsilon$, which is quite good and as expected from both algorithms (it is easy

TABLE 1
Accuracy of singular values from LINPACK SVD.

Class	% all digits	% 12–14 digits	% 8–12 digits	% 4–8 digits	% 0–4 digits	% −1 digits	% nonzero, no digits	% zero, no digits
1	100	0	0	0	0	0	0	0
2	42	14	11	0	29	1	0	3
3	99.5	.5	0	0	0	0	0	0
4*	59	11	4	2	21	1	0	2
5	94	0	0	0	0	0	0	6
6	94	0	0	0	0	0	0	6
7	90	1	.5	0	.5	0	1	7
8	80.5	4.5	5	3	1	.5	.5	5
9	80	6.5	7	2.5	1	0	0	3
10	91	4.5	3	1.5	0	0	0	0
11	98	2	0	0	0	0	0	0
12	100	0	0	0	0	0	0	0

* The algorithm did not converge for one of the test matrices (this matrix was not counted in computing the percentages).

to show the convergence criteria for both algorithms leave the residual near the roundoff level). We do not yet have a complete perturbation theory or better accuracy tests for the singular vectors (see § 9 for further discussion).

Table 2 provides a measure of the difficulty of the different problem classes that is independent of matrix dimension. The usual rule of thumb for the number of QR sweeps it takes to compute the SVD is two sweeps per singular value [15, p. 165]. If convergence always takes place at the end of the matrix, this means there will be two sweeps on a matrix of length i, for $i = n, n-1, \cdots, 3$ (two-by-two matrices are handled specially). Here, n is the dimension of the original matrix. Thus counting one QR sweep on a matrix of length i as i "QR inner loops," we expect an average of about $n(n+1)$ "QR inner loops" for the entire SVD. Thus the quantity "QR inner loops" divided by $n(n+1)/2$ should be a measure of the difficulty of computing the SVD of a matrix that is independent of dimension, and we expect it to equal two on the average. For each of the twelve problem classes, and for the three algorithms old SVD (LINPACK), new SVD without singular vectors, and new SVD with singular vectors, the minimum, average, and maximum of the quantity "QR inner loops" divided by $n(n+1)/2$ are given in Table 2. Recall that we use different convergence criteria depending on whether or not we compute singular vectors, which is why we have different columns for these two cases.

TABLE 2
QR *Inner Loops*$/(n(n+1)/2)$ *for old and new* SVD *algorithms with tol* $= 100\varepsilon \approx 10^{-14}$.

	Old SVD			New SVD without vectors			New SVD with vectors		
Class	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
1	.60	.90	1.33	.09	.36	.91	.09	.49	1.11
2	.60	1.94	3.07	.09	.36	.91	.09	.49	1.11
3	.61	.85	1.19	.56	.82	1.19	.56	.82	1.19
4	.32	1.04	1.80	.31	.58	1.00	.35	.60	1.04
5	.07	.45	1.11	.09	.54	1.29	.09	.57	1.42
6	.07	.40	.93	.09	.54	1.29	.09	.57	1.42
7	.10	1.32	2.31	.10	1.04	1.85	.10	1.04	1.85
8	.41	.64	.95	.25	.47	.75	.26	.49	.77
9	.79	.94	1.29	.51	.73	.89	.57	.75	.93
10	1.07	1.29	1.57	.98	1.19	1.47	1.04	1.22	1.48
11	1.97	2.26	2.52	2.07	2.20	2.38	2.06	2.20	2.41
12	1.53	1.53	1.53	2.96	2.96	2.96	2.96	2.96	2.96

It is interesting to note in Table 2 that only in Class 11 is our expectation of two QR sweeps per singular value for the standard SVD nearly fulfilled. Recall that Class 11 has matrices all of whose entries are uniformly distributed between $\pm .5$. Otherwise, either the average is much lower or there is a great variability in the number of QR sweeps needed (Class 2). The same comments hold for the new algorithm, except for Class 12, which was chosen to make the new algorithm look as bad as possible. Even so, it is within a factor of two of the old algorithm.

Table 3 gives timing comparisons between the old and new algorithms. The results depend on whether singular vectors are computed (Job = v in Table 3) or not (Job = nv). There were several statistics collected. First, the number of QR inner loops for each algorithm was counted, and the ratio of QR inner loops for the new algorithm to QR inner loops for the old algorithm computed; these statistics (minimum, average, and maximum ratios, the same for the other statistics) are shown in columns 3–5 of Table

TABLE 3
Timing comparisons of old and new SVD *algorithms with tol* = $100\varepsilon \approx 10^{-14}$.

		Ratio of inner loops New SVD/Old SVD			Ratio of times (with bidiagonalization) New SVD/Old SVD			Ratio of times (without bidiagonalization) New SVD/Old SVD		
Class	Job	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
1	nv	.15	.37	.77	.69	.77	.85	.29	.41	.63
	v	.15	.48	.84	.67	.75	.87	.26	.49	.77
2	nv	.10	.18	.34	.37	.58	.82	.14	.23	.37
	v	.10	.23	.36	.37	.51	.66	.13	.26	.34
3	nv	.86	.96	1.02	.91	.94	.96	.73	.77	.80
	v	.86	.96	1.03	.95	.97	1.01	.88	.95	1.03
4	nv	.44	.60	.99	.72	.85	1.02	.40	.58	1.15
	v	.44	.63	1.10	.69	.78	1.03	.46	.63	1.10
5	nv	.67	1.23	2.00	.97	1.06	1.12	.94	1.76	3.42
	v	.67	1.26	2.00	1.01	1.09	1.16	1.06	1.33	1.77
6	nv	.67	1.29	2.00	1.03	1.07	1.11	1.11	1.72	3.16
	v	.67	1.33	2.00	.99	1.11	1.24	.98	1.33	1.56
7	nv	.10	.80	1.00	.51	.91	1.12	.16	1.03	3.32
	v	.10	.80	1.00	.44	.89	1.14	.15	.93	2.04
8	nv	.38	.79	1.61	.81	.92	1.07	.45	.75	1.32
	v	.40	.82	1.64	.74	.91	1.20	.47	.82	1.47
9	nv	.52	.78	.97	.78	.89	.96	.48	.69	.87
	v	.52	.81	1.02	.72	.89	1.01	.54	.80	1.00
10	nv	.62	.94	1.19	.78	.90	.98	.53	.77	.94
	v	.67	.96	1.21	.76	.95	1.09	.63	.93	1.18
11	nv	.89	.98	1.12	.88	.93	1.00	.79	.87	.99
	v	.86	.98	1.13	.87	.96	1.06	.83	.94	1.09
12	nv	1.93	1.93	1.93	1.13	1.13	1.13	1.37	1.37	1.37
	v	1.93	1.93	1.93	1.55	1.55	1.55	1.87	1.87	1.87

3. The timings also depend on whether or not we count the time to bidiagonalize. The time to bidiagonalize is quite large and can swamp the second, iterative part. Therefore we computed timing ratios (new algorithm to old algorithm) both with and without the initial bidiagonalization. The identical bidiagonalization code was used for the old and new algorithms. We performed the bidiagonalization part of the algorithm on a different, dense matrix, so that the algorithm and floating point hardware would not recognize they were dealing with a bidiagonal input matrix and so bypass some of the work. Columns 6–8 of Table 3 include the bidiagonalization phase, and columns 9–11 exclude it.

Whenever a number less than 1 appears in the table, it means the new algorithm was faster, and numbers greater than 1 indicate the old algorithm was faster. An examination of the table shows that on the whole the performance of the two algorithms is comparable. Counting bidiagonalization, the new algorithm varies from over 2.7 times faster (Class 2) to 1.55 times slower (Class 12). Not counting bidiagonalization the extremes are 7.7 times faster to 3.42 times slower; the extra overhead of bidiagonalization moderates the extremes. On simply graded matrices (Classes 1–4) and on random matrices (Classes 8–11) the new algorithm always did better than the old on the average. With the diagonal and off-diagonal being graded differently (Classes 5–6), the old algorithm was generally a little faster. In Classes 5–7 the largest ratios occurred in examples where convergence was very fast with both algorithms, the old SVD's faster convergence criterion winning out over the new algorithm's more careful but more expensive convergence test. In Class 7 without bidiagonalization and without computing singular vectors, there were only two matrices where the old algorithm beat the new (by factors of 2.42 and 3.32); in both cases both algorithms converged after a *single* QR sweep. Thus the difference in times can be attributed to the slower convergence criteria of the new algorithm; in both cases convergence was nearly immediate. Similarly, in Classes 5 and 6 without bidiagonalization and without computing singular vectors, whenever the old algorithm beat the new algorithm by more than 32 percent, the ratio "QR inner loops"$/(n(n+1)/2)$ was less than .17. In Class 7 and many examples in Classes 5-6 there were generally a few very small singular values and the rest large and evenly spaced over a range of at most a few factors of 10; the new algorithm deflated out the smallest singular values after 1 or 2 sweeps and spent the rest of the time working on the closely spaced singular values. It appears our criterion for choosing between zero and nonzero shift chooses the zero shift quite often, sometimes sacrificing cubic convergence until many singular values have been deflated. The single matrix in Class 12 was therefore chosen with very closely spaced singular values in order to make the new algorithm perform as poorly as possible; in this example the average number of (mostly zero shift) QR sweeps per singular value was 2.96 for the new algorithm, whereas the average number of (shifted) QR sweeps per singular value was 1.53 for the old algorithm, which still computed them all correctly. We are not currently able to find another criterion permitting more frequent nonzero shifts while still guaranteeing high relative accuracy. Nonzero shifts for fairly small singular values frequently do not cause inaccuracy in practice because small rotation angles prevent mixing large and small magnitude matrix entries; unfortunately this phenomenon seems hard to exploit systematically.

From Table 3, it appears that Convergence Criteria 2a and 2b are not very effective, since the ratio of inner loops (columns 3-5) does not change very much when Job = nv (singular vectors are not computed and Criteria 2a and 2b are used) and when Job = v (singular vectors are computed and Criteria 2a and 2b are *not* used). This is somewhat misleading, however. Closer inspection of the test cases shows that in Classes 1 and 2, Criteria 2a and 2b cut the ratio of inner loops in half for matrices that have constant ratios between adjacent diagonal matrix entries. In these cases, the algorithm converges in a single QR sweep, instead of two QR sweeps. But for the other test matrices in Classes 1 and 2, where matrix entries come in equal pairs, Criteria 2a and 2b have no effect at all. The excellent performance on the first set of test matrices is watered down in the statistics presented. Of course, since this speedup is only for matrices for which the algorithm is already quite fast, we could simply omit Criteria 2a and 2b altogether; this would have the advantage of computing identical singular values independent of whether we also compute singular vectors.

Another interesting feature of Table 3 is the difference between Classes 1 and 2. Recall that these matrices differ only in the order of the data. In Class 1, the old and new algorithms are always chasing the bulge in the same direction; in Class 2 they always chase the bulge in the opposite direction, which degrades the accuracy of the old algorithm as mentioned above. It also degrades the performance by about a factor of 2: in Class 1 (without bidiagonalization and without computing singular vectors) the new algorithm is about twice as fast as the old on the average, and in Class 2 four times as fast.

We next present some timings for our algorithm with $tol = 10^{14}\varepsilon \approx 10^{-2}$ compared to the new algorithm with $tol = 100\varepsilon \approx 10^{-14}$. This low accuracy requirement speeds up the algorithm while still providing order-of-magnitude correct singular values; thus it may be of use for rank determination. Only "Job = nv" (singular values only) cases were run. The new algorithm with $tol = 10^{14}\varepsilon$ was always faster than the new algorithm with $tol = 10^2\varepsilon$ except for two matrices in Class 4 and one in Class 5. In all cases the computed singular values were good to at least two figures as expected. (See Table 4.)

As mentioned at the end of § 4 on convergence criteria, we may use the much less stringent criteria (4.5) and (4.6) if only absolute accuracy rather than relative accuracy in the singular values is desired. In Table 5 we show timing comparisons between the new algorithm where each singular value is computed to an absolute accuracy of $tol \cdot \|A\| = 100\varepsilon\|A\| \approx 10^{-14} \cdot \|A\|$, and the new algorithm with a relative accuracy tolerance $tol = 100\varepsilon \approx 10^{-14}$ as in Table 3. The format is the same as in Table 3. As can be seen from Table 5, the absolute convergence criterion almost always leads to faster convergence than the relative convergence criterion.

TABLE 4
Timing comparisons of new SVD *algorithm with* $tol = 10^{14}\varepsilon \approx 10^{-2}$ *and* $tol = 10^2\varepsilon \approx 10^{-14}$.

	Ratio of inner loops SVD ($tol \approx 10^{-2}$)/ SVD ($tol \approx 10^{-14}$)			Ratio of times (with bidiagonalization) SVD ($tol \approx 10^{-2}$)/ SVD ($tol \approx 10^{-14}$)			Ratio of times (without bidiagonalization) SVD ($tol \approx 10^{-2}$)/ SVD ($tol \approx 10^{-14}$)		
Class	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
1	.19	.58	1.00	.80	.91	1.00	.33	.65	.95
2	.19	.58	1.00	.79	.91	1.00	.32	.66	.95
3	.47	.70	.84	.88	.95	.98	.68	.81	.92
4	.56	.86	1.05	.94	.99	1.08	.71	.93	1.24
5	.07	.35	1.00	.61	.85	1.00	.19	.54	1.00
6	.07	.35	1.00	.61	.85	1.00	.19	.53	1.00
7	.09	.33	1.00	.46	.70	1.00	.13	.37	1.00
8	.10	.23	.46	.80	.88	.96	.26	.45	.71
9	.07	.20	.40	.77	.83	.91	.21	.36	.57
10	.04	.17	.30	.68	.74	.81	.15	.29	.44
11	.27	.41	.48	.63	.71	.76	.31	.46	.53
12	.14	.14	.14	.68	.68	.68	.21	.21	.21

Finally, we discuss the implications of our results for the "perfect shift" strategy for computing singular vectors (or eigenvectors). This strategy advocates computing the singular values (or eigenvalues) by the quickest available method without accumulating singular vectors, and then using these computed singular values as "perfect shifts" in the QR iteration to compute the singular vectors in one or possibly two QR sweeps. The hope is that by avoiding the work of accumulating vectors while converging to accurate singular values, time will be saved by computing the singular vectors afterwards in one or two sweeps each. Unfortunately, our numerical results indicate this approach will not work in general. For when our hybrid algorithm chooses to do an implicit zero shift, it is in fact doing a perfect shift within the limits of roundoff error. Depending on the distribution of singular values, this can take more or less time to converge. Therefore we cannot assume one or two sweeps with the "perfect shift" will result in converged singular vectors, and we could well end up doing as many sweeps to compute the singular vectors as the singular values. This will not happen in general, and a clever algorithm might be able to decide when perfect shifts are useful and then use them, perhaps by keeping track of which deflated subblocks of the matrix do not require zero shifts and using the perfect shift strategy on them.

8. Detailed error analysis. In this section we present a detailed error analysis of the implicit zero-shift QR algorithm (Theorems 6 and 7). We begin with the assumptions

TABLE 5
Timing comparisons with absolute accuracy $tol = 100\varepsilon \cdot \|A\| \approx 10^{-14}\|A\|$ *versus relative accuracy* $tol = 100\varepsilon \approx 10^{-14}$.

		Ratio of inner loops SVD (absolute *tol*)/ SVD (relative *tol*)			Ratio of times (with bidiagonalization) SVD (absolute *tol*)/ SVD (relative *tol*)			Ratio of times (without bidiagonalization) SVD (absolute *tol*)/ SVD (relative *tol*)		
Class	Job	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
1	nv	.21	.48	1.00	.87	.91	.97	.32	.51	.92
	v	.17	.40	.77	.76	.83	.89	.32	.47	.74
2	nv	.21	.48	1.00	.87	.91	.97	.32	.52	.92
	v	.17	.40	.77	.76	.83	.89	.31	.46	.75
3	nv	.06	.39	.87	.81	.88	.97	.13	.42	.85
	v	.06	.39	.86	.59	.73	.92	.09	.41	.85
4	nv	.03	.25	.70	.82	.88	.94	.11	.30	.71
	v	.03	.25	.73	.66	.73	.85	.07	.28	.72
5	nv	.14	.47	.92	.75	.89	.94	.21	.47	.87
	v	.14	.46	.86	.68	.85	.91	.26	.50	.85
6	nv	.14	.47	.92	.75	.89	.94	.21	.47	.87
	v	.14	.46	.86	.68	.85	.91	.25	.50	.85
7	nv	.82	.94	1.00	.92	.96	.99	.41	.83	.98
	v	.82	.94	1.00	.92	.96	1.00	.66	.88	1.00
8	nv	.26	.40	.64	.82	.88	.93	.32	.43	.55
	v	.25	.40	.62	.66	.78	.89	.29	.43	.62
9	nv	.50	.68	.95	.88	.91	.97	.54	.67	.87
	v	.51	.68	.93	.78	.85	.96	.53	.68	.91
10	nv	.60	.83	1.00	.88	.93	.98	.65	.82	.96
	v	.61	.83	.97	.78	.90	.97	.62	.83	.95
11	nv	.97	1.00	1.00	1.00	1.01	1.02	.99	1.01	1.04
	v	.97	1.00	1.05	.98	1.00	1.04	.97	1.00	1.05
12	nv	.88	.88	.88	.95	.95	.95	.89	.89	.89
	v	.88	.88	.88	.91	.91	.91	.87	.87	.87

used in the error analysis. Our model of error in floating point arithmetic was given above in (3.5). It implies that overflow and underflow do not occur; we discuss susceptibility to overflow and underflow briefly at the end. In our analysis εs with numerical subscripts denote quantities bounded in magnitude by ε, where ε is the machine precision. Our analysis will be linearized in the sense that we will replace quantities such as $(1+\varepsilon_1)*(1+\varepsilon_2)$ by $1+(\varepsilon_1+\varepsilon_2)$ and $(1+\varepsilon_1)/(1+\varepsilon_2)$ by $1+(\varepsilon_1-\varepsilon_2)$; such approximations can be made rigorous by assuming all $n\varepsilon_i$ are less than .1 in magnitude and increasing the final error bound by a factor 1.06 [18, p. 113].

LEMMA 5. *Let* $\cos\theta$, $\sin\theta$, *and* ρ *denote the exact outputs of ROT for inputs f and g and exact arithmetic. Now consider the floating point version of ROT applied to the perturbed inputs* $\hat{f} = f(1+\varepsilon_f)$ *and* $\hat{g} = g(1+\varepsilon_g)$, *and let* $cs = (1+\varepsilon_{cs})\cos\theta$, $sn = (1+\varepsilon_{sn})\sin\theta$, *and* $r = (1+\varepsilon_r)\rho$ *denote the computed results, where we assume neither overflow nor underflow occurs. Then we may estimate the relative errors* ε_{cs}, ε_{sn}, *and* ε_r *as follows*:

$$\varepsilon_{cs} = (\varepsilon_f - \varepsilon_g)\sin^2\theta + \varepsilon'_{cs} \quad \text{where } |\varepsilon'_{cs}| \leqq \tfrac{21}{4}\varepsilon,$$

$$\varepsilon_{sn} = (\varepsilon_g - \varepsilon_f)\cos^2\theta + \varepsilon'_{sn} \quad \text{where } |\varepsilon'_{sn}| \leqq \tfrac{21}{4}\varepsilon,$$

$$\varepsilon_r = \varepsilon_g \sin^2\theta + \varepsilon_f\cos^2\theta + \varepsilon'_r \quad \text{where } |\varepsilon'_r| \leqq \tfrac{13}{4}\varepsilon.$$

Proof. We only consider the case $|\hat{f}|>|\hat{g}|$; the other case is analogous. In the following ε's with numeric subscripts indicate quantities bounded by ε which may be functions of previous ε_i's as described in the first paragraph of this section. Then applying (3.5) systematically to the expressions in *ROT*, and using the fact that $|\hat{g}/\hat{f}|<1$, we obtain

$$\begin{aligned}
t&=\frac{g}{f}(1+\varepsilon_g-\varepsilon_f+\varepsilon_1),\\
tt&=(1+\varepsilon_4)\cdot[(1+\varepsilon_3)\cdot(1+t^2(1+\varepsilon_2))]^{1/2}\\
&=(1+7\varepsilon_5/4)[1+t^2]^{1/2}\\
&=(1+7\varepsilon_5/4)[1+(g/f)^2(1+2(\varepsilon_g-\varepsilon_f+\varepsilon_1))]^{1/2}\\
&=(1+7\varepsilon_5/4)(1+(\varepsilon_g-\varepsilon_f+\varepsilon_1)(g/f)^2/(1+(g/f)^2))[1+(g/f)^2]^{1/2}\\
&=(1+9\varepsilon_6/4+(\varepsilon_g-\varepsilon_f)\sin^2\theta)\sec\theta,\\
cs&=(1+\varepsilon_7)/tt=(1+13\varepsilon_8/4+(\varepsilon_f-\varepsilon_g)\sin^2\theta)\cos\theta,\\
sn&=(1+\varepsilon_9)t\cdot cs=(1+21\varepsilon_{10}/4+(\varepsilon_g-\varepsilon_f)\cos^2\theta)\sin\theta,\\
r&=(1+\varepsilon_{11})f\cdot tt=(1+13\varepsilon_{12}/4+\varepsilon_g\sin^2\theta+\varepsilon_f\cos^2\theta)\rho.
\end{aligned}$$
□

To analyze the errors in the implicit zero-shift QR algorithm, we need to investigate how the errors accumulate from one pass through the loop to the next. It turns out the errors in f and *oldcs* are the essential ones.

LEMMA 6. *Let f_i and $oldcs_i$ denote the true values of f and oldcs at the entry to the ith iteration of the loop in the implicit zero-shift* QR *algorithm. Let θ_{1i} and θ_{2i} be the true values of the two rotation angles in the ith iteration of the loop. In other words, f_i, $oldcs_i$, θ_{1i}, and θ_{2i} are the values that would have been computed had all arithmetic been exact. Let $f_i(1+\varepsilon_{fi})$ and $oldcs_i(1+\varepsilon_{oldcs_i})$ denote the actual floating point values of f and oldcs at the top of the loop, with all previous loop iterations having been done in floating point without any overflows or underflows. Then*

$$\begin{bmatrix}|\varepsilon_{f_{i+1}}|\\ |\varepsilon_{oldcs_{i+1}}|\end{bmatrix}\leqq\begin{bmatrix}\sin^2\theta_{1i} & 0\\ 2\cos^2\theta_{1i}\cdot\sin^2\theta_{2i} & \sin^2\theta_{2i}\end{bmatrix}\cdot\begin{bmatrix}|\varepsilon_{fi}|\\ |\varepsilon_{oldcs_i}|\end{bmatrix}+\begin{bmatrix}25\varepsilon/4\\ 21\varepsilon/4+21\varepsilon\cdot\sin^2\theta_{2i}/2\end{bmatrix}. \tag{8.1}$$

In terms of these expressions, we can bound the errors in the computed values of e_i and s_i:

$$e_i=\text{“}true\ e_i\text{”}\cdot(1+\varepsilon_{e_i})\quad and\quad s_i=\text{“}true\ s_i\text{”}\cdot(1+\varepsilon_{s_i})$$

where

$$\varepsilon_{e_i}=-\varepsilon_{oldcs_i}\cdot\cos^2\theta_{2i}-2\varepsilon_{fi}\cdot\cos^2\theta_{1i}\cdot\cos^2\theta_{2i}+\varepsilon_{f_{i+1}}\cdot cos^2\theta_{1,i+1}+20\varepsilon \tag{8.2}$$

and

$$\varepsilon_{s_i}=\varepsilon_{fi}\cdot\cos^2\theta_{1i}\cos 2\theta_{2i}+\varepsilon_{oldcs_i}\cdot\cos^2\theta_{2i}+\tfrac{15}{2}\varepsilon_{14}+\tfrac{25}{4}\sin^2\theta_{2i}\varepsilon_{15}. \tag{8.3}$$

ε_{e_i} and e_{s_i} may be further bounded by

$$|\varepsilon_{e_i}|\leqq|\varepsilon_{oldcs_i}|+2|\varepsilon_{fi}|+|\varepsilon_{f_{i+1}}|+20\varepsilon \tag{8.4}$$

and

$$|\varepsilon_{s_i}|\leqq|\varepsilon_{fi}|+|\varepsilon_{oldcs_i}|+25\varepsilon\sin^2\theta_{2i}/4+15\varepsilon/2. \tag{8.5}$$

Proof. We apply (3.5) and Lemma 5 systematically to the expressions in the algorithm. As before, ε's with numeric subscripts denote expressions bounded in magnitude by ε.

Note that at the top of the loop, g is known exactly. Therefore, after the first call to *ROT* we have

$$cs = \cos\theta_{1i}\cdot(1+\varepsilon_{fi}\cdot\sin^2\theta_{1i}+\tfrac{21}{4}\varepsilon_1),$$
$$sn = \sin\theta_{1i}\cdot(1-\varepsilon_{fi}\cdot\cos^2\theta_{1i}+\tfrac{21}{4}\varepsilon_2),$$
$$r = \text{“}true\ r\text{”}\cdot(1+\varepsilon_{fi}\cdot\cos^2\theta_{1i}+\tfrac{13}{4}\varepsilon_3).$$

The errors in f, g, and h are given by

$$f=(1+\varepsilon_4)\cdot oldcs\cdot r = \text{“}true\ f\text{”}\cdot(1+\varepsilon_{oldcs_i}+\varepsilon_{fi}\cdot\cos^2\theta_{1i}+\tfrac{17}{4}\varepsilon_5),$$
$$g=(1+\varepsilon_6)\cdot s_{i+1}\cdot sn = \text{“}true\ g\text{”}\cdot(1-\varepsilon_{fi}\cdot\cos^2\theta_{1i}+\tfrac{25}{4}\varepsilon_7),$$
$$h=(1+\varepsilon_8)\cdot s_{i+1}\cdot cs = \text{“}true\ h\text{”}\cdot(1+\varepsilon_{fi}\cdot\sin^2\theta_{1i}+\tfrac{25}{4}\varepsilon_9).$$

After the second call to *ROT* we have

$$cs = \cos\theta_{2i}\cdot(1+\varepsilon_{oldcs_i}\cdot\sin^2\theta_{2i}+2\varepsilon_{fi}\cdot\cos^2\theta_{1i}\cdot\sin^2\theta_{2i}+\tfrac{21}{2}\varepsilon_{10}\cdot\sin^2\theta_{2i}+\tfrac{21}{4}\varepsilon_{11}),$$
$$sn = \sin\theta_{2i}\cdot(1-\varepsilon_{oldcs_i}\cdot\cos^2\theta_{2i}-2\varepsilon_{fi}\cdot\cos^2\theta_{1i}\cdot\cos^2\theta_{2i}+\tfrac{21}{2}\varepsilon_{12}\cdot\cos^2\theta_{2i}+\tfrac{21}{4}\varepsilon_{13}),$$
$$r = \text{“}true\ r\text{”}\cdot(1+\varepsilon_{fi}\cdot\cos^2\theta_{1i}\cdot\cos 2\theta_{2i}+\varepsilon_{oldcs_i}\cdot\cos^2\theta_{2i}+\tfrac{15}{2}\varepsilon_{14}+\tfrac{25}{4}\varepsilon_{15}\cdot\sin^2\theta_{2i}).$$

Since $oldcs = cs$ and $f = h$ at the bottom of the loop, we have shown that at the start of the next loop

$$f = f_{i+1}(1+\varepsilon_{fi}\cdot\sin^2\theta_{1i}+\tfrac{25}{4}\varepsilon_9),$$
$$oldcs = oldcs_{i+1}(1+\varepsilon_{oldcs_i}\cdot\sin^2\theta_{2i}+2\varepsilon_{fi}\cdot\cos^2\theta_{1i}\cdot\sin^2\theta_{2i}+\tfrac{21}{2}\varepsilon_{10}\cdot\sin^2\theta_{2i}+\tfrac{21}{4}\varepsilon_{11})$$

as desired. The expressions for the errors in e_i and s_i follow from plugging the above bounds into the expressions $e_{i-1} = oldsn * r$ and $s_i = r$. □

From (8.4) and (8.5) we see that the errors in the computed e_i and s_i are governed by the errors e_{fi} and ε_{oldcs_i}, and that the growths of these errors are governed by the recurrence (8.1). A simple but somewhat pessimistic bound on these errors is given by the following lemma.

LEMMA 7. *Let* ε_{fi}, ε_{oldcs_i}, ε_{e_i}, *and* ε_{s_i} *be as in Lemma* 6. *Then*

$$|\varepsilon_{fi}| \leqq 25(i-1)\varepsilon/4,$$
$$|\varepsilon_{oldcs_i}| \leqq 113(i-1)\varepsilon/4,$$
$$|\varepsilon_{e_i}| \leqq (138i-53)\varepsilon/4,$$
$$|\varepsilon_{s_i}| \leqq (113i-58)\varepsilon/4.$$

Proof. Replace the recurrence (8.1) by

$$E_{i+1} = A_i\cdot E_i + F_i$$

where

$$A_i = \begin{bmatrix} \sin^2\theta_{1i} & 0 \\ 2\cos^2\theta_{1i}\cdot\sin^2\theta_{2i} & \sin^2\theta_{2i}\end{bmatrix},\qquad F_i = \begin{bmatrix} 25\varepsilon/4 \\ 21\varepsilon/4 + 21\varepsilon\cdot\sin^2\theta_{2i}/2\end{bmatrix}$$

and the entries of E_i are upper bounds for $|e_{fi}|$ and $|\varepsilon_{oldcs_i}|$. Taking $E_1 = 0$, this recurrence has the solution

$$E_{i+1} = \sum_{j=1}^{i} \left(\sum_{k=j+1}^{i} A_k \right) F_j. \tag{8.6}$$

Trivial bounds for A_i and F_i are

$$|A_i| \leqq \begin{bmatrix} \sin^2\theta_{1i} & 0 \\ 2\cos^2\theta_{1i} & 1 \end{bmatrix} \quad \text{and} \quad |F_i| \leqq \begin{bmatrix} 25\varepsilon/4 \\ 63\varepsilon/4 \end{bmatrix}. \tag{8.7}$$

A simple induction shows that

$$\prod_{k=j+1}^{i} \begin{bmatrix} \sin^2\theta_{1k} & 0 \\ 2\cos^2\theta_{1k} & 1 \end{bmatrix} = \begin{bmatrix} \prod_{k=j+1}^{i} \sin^2\theta_{1k} & 0 \\ 2\left(1 - \prod_{k=j+1}^{i} \sin^2\theta_{1k}\right) & 1 \end{bmatrix}$$

and the rest of the proof is a straightforward computation. □

In other words, the relative errors in the computed e_i and s_i are bounded by a linear function of i, and so the largest relative error is bounded by a linear function of the matrix dimension n. We can now apply Theorem 2 of § 2 to bound the errors in the singular values of the transformed matrix.

THEOREM 6. *Let B be an n-by-n bidiagonal matrix and B' the matrix obtained by running the implicit zero-shift* QR *algorithm on B. Let the singular values of B be $\sigma_1 \geqq \cdots \geqq \sigma_n$, and the singular values of B' be $\sigma'_1 \geqq \cdots \geqq \sigma'_n$. Then if*

$$\omega \equiv 69n^2\varepsilon < 1, \tag{8.8}$$

the relative differences between the singular values of B and the singular values of B' are bounded as follows:

$$|\sigma_i - \sigma'_i| \leqq \frac{\omega}{1-\omega}\sigma_i.$$

Let B_k be the matrix obtained after k repetitions of the implicit zero-shift QR *algorithm, and let $\sigma_{k1} \geqq \cdots \geqq \sigma_{kn}$ be its singular values. Then if condition* (8.8) *holds we have*

$$|\sigma_i - \sigma_{ki}| \leqq \left(\frac{1}{(1-\omega)^k} - 1 \right) \cdot \sigma_i \approx 69kn^2\varepsilon \cdot \sigma_i,$$

where the approximation to the last upper bound holds if $k\omega \ll 1$.

Proof. Plug the bounds of Lemma 7 into Theorem 2. □

Actually, Lemma 7 and Theorem 6 are quite pessimistic, since the upper bounds in (8.7) are unattainable. In fact, as we approach convergence, we expect the rotation angles θ_{1i} and θ_{2i} to approach zero, which means the matrix A_i should approach zero. We can use this fact to obtain a much stronger error bound in the region of convergence.

LEMMA 8. *Let ε_{fi}, ε_{oldcs_i}, ε_{e_i}, and ε_{s_i} be as in Lemma 6. Assume further that all the rotation angles θ during the course of the algorithm satisfy $\sin^2\theta \leqq \tau < 1$. Then*

$$|\varepsilon_{fi}| \leqq \frac{25\varepsilon}{4(1-\tau)},$$

$$|\varepsilon_{oldcs_i}| \leqq \frac{50\tau\varepsilon}{4(1-\tau)^2} + \frac{21\tau\varepsilon}{2(1-\tau)} + \frac{21\varepsilon}{4(1-\tau)},$$

$$|\varepsilon_{e_i}| \leqq \frac{50\tau\varepsilon}{4(1-\tau)^2} + \frac{21\tau\varepsilon}{2(1-\tau)} + \frac{24\varepsilon}{1-\tau} + 20\varepsilon,$$

$$|\varepsilon_{s_i}| \leqq \frac{50\tau\varepsilon}{4(1-\tau)^2} + \frac{21\tau\varepsilon}{2(1-\tau)} + \frac{23\varepsilon}{2(1-\tau)} + \frac{25\tau\varepsilon}{4} + \frac{15\varepsilon}{2}.$$

Proof. Use the bounds

$$|A_i| \leqq \begin{bmatrix} \tau & 0 \\ 2\tau & \tau \end{bmatrix} \quad \text{and} \quad |F_i| \leqq \begin{bmatrix} 25\varepsilon/4 \\ 21\varepsilon/4 + 21\varepsilon\tau/2 \end{bmatrix}.$$

The rest of the proof is a straightforward computation from (8.6). □

These bounds permit us state the following improvement to Theorem 6.

THEOREM 7. *Let B be an n-by-n bidiagonal matrix and B′ the matrix obtained by running the implicit zero-shift* QR *algorithm on B. Assume that all the rotation angles θ during the course of the algorithm satisfy* $\sin^2 \theta \leqq \tau < 1$. *Let the singular values of B be* $\sigma_1 \geqq \cdots \geqq \sigma_n$, *and the singular values of B′ be* $\sigma'_1 \geqq \cdots \geqq \sigma'_n$. *Then if*

$$\omega \equiv \frac{88n\varepsilon}{(1-\tau)^2} < 1, \tag{8.9}$$

the relative differences between the singular values of B and the singular values of B′ are bounded as follows:

$$|\sigma_i - \sigma'_i| \leqq \frac{\omega}{1-\omega}\sigma_i.$$

Let B_k *be the matrix obtained after k repetitions of the implicit zero-shift* QR *algorithm, where we assume all rotation angles θ satisfy* $\sin^2\theta \leqq \tau < 1$. *Let* $\sigma_{k1} \geqq \cdots \geqq \sigma_{kn}$ *be the singular values of* B_k. *Then if condition* (8.9) *holds we have*

$$|\sigma_i - \sigma_{ki}| \leqq \left(\frac{1}{(1-\omega)^k} - 1\right) \cdot \sigma_i \approx \frac{88kn\varepsilon}{(1-\tau)^2} \cdot \sigma_i,$$

where the approximation to the last upper bound holds if $k\omega \ll 1$.

Proof. Plug the bounds of Lemma 8 into Theorem 2. □

Thus if the rotation angles are all bounded away from $\pi/2$, the error after k iterations of the implicit zero-shift QR algorithm can grow essentially only as the product kn. The algorithm can easily compute τ as it proceeds, and so compute its own error bound if desired. In the numerical experiments in § 7, we observed no error growth at all, and so as is often the case an algorithm behaves much better in practice than rigorous error bounds can guarantee.

Now we briefly consider over/underflow. Most of the error analysis presented here can be extended to take over/underflow into account. Techniques for error analysis in the presence of underflow are discussed in [2]. If over/underflow is handled as suggested in the IEEE Floating Point Standard [10], then using Sylvester's theorem to count the number of eigenvalues less than x (2.1) can be made completely impervious to over/underflow [12]: If some $d_i = \pm 0$, then $d_{i+1} = \pm\infty$ and $d_{i+2} = a_{i+2}$, and we count the number of d_i whose sign bit is negative (i.e., including -0 and $-\infty$). Rules for arithmetic with ± 0 and $\pm\infty$ are described in detail in [10].

9. The accuracy of the computed singular vectors.

In this section we assess the accuracy of the computed singular vectors. Just as with the standard SVD, the new algorithm guarantees a small residual in the sense that both $\|B\hat{v} - \hat{\sigma}\hat{u}\|$ and $\|B^T\hat{u} - \hat{\sigma}\hat{v}\|$ are on the order of $\varepsilon\|B\|$, where $\hat{\sigma}$ is the computed singular value and $\hat{u}$ and $\hat{v}$ are the computed singular vectors. However, in contrast to the singular values, high relative accuracy in the bidiagonal matrix entries does not guarantee high relative accuracy in the singular vectors; we will give a 2-by-2 example to illustrate this. It also turns out to be impossible to guarantee a tiny componentwise relative backwards error, where each computed singular vector of B would be the exact singular vector of a small componentwise relative perturbation $B+\delta B$ of B, with $|\delta B| \leqq \eta|B|$, η on the order of machine precision. We will also demonstrate this with a small example.

In place of such simple a priori forward or backward error bounds, our bounds will depend on the singular value distribution. Briefly, the closer together singular values are, the less accurately their corresponding singular vectors can be computed. This dependency is captured in the well-known "gap" theorem [15, p. 222], which can be used to show that the angular error in the computed singular vectors corresponding to σ_i is bounded by the largest roundoff error committed divided by the "gap" or difference between σ_i and its nearest neighbor $\sigma_{i\pm 1}$. This well-known bound holds for the standard SVD applied to dense matrices as well as the new algorithm.

Numerical experience leads us to make the following conjecture for the new algorithm applied to bidiagonal matrices, which would significantly strengthen the bound in the last paragraph: the "gap" $\min|\sigma_i - \sigma_{i\pm 1}|$ in the denominator of the above error bound can be replaced by the "relative gap" $\min(|\sigma_i - \sigma_{i\pm 1}|/\sigma_i)$. Since the relative gap can be much larger than the gap, the resulting error bound can be much smaller. For example, if B is 3-by-3 bidiagonal matrix with singular values $\sigma_1 = 1$, $\sigma_2 = 2 \cdot 10^{-20}$, and $\sigma_3 = 10^{-20}$, the old error bounds for the vectors corresponding to the two tiny singular values are on the order of $10^{20}\varepsilon$ since the gap is 10^{-20}. However, the conjectured bounds are both ε since the relative gap between $2 \cdot 10^{-20}$ and 10^{-20} is 1. Proving this conjecture rigorously remains an open problem, although a supporting result appears in [1].

Now we present a 2-by-2 example showing that small relative perturbations in the entries of a bidiagonal matrix can cause large perturbations in the singular vectors:

$$A(\eta) = \begin{bmatrix} 1+\eta & \varepsilon \\ 0 & 1 \end{bmatrix}.$$

As η varies from 0 to ε, an easy computation shows that both left and right singular vectors rotate by 22.5 degrees.

The same example can be used to show that no tiny componentwise relative backward error bound can hold. Specifically, let $\hat{u}_i$ and $\hat{v}_i$ be the left and right unit singular vectors of

$$A = \begin{bmatrix} 1+\varepsilon & \varepsilon \\ 0 & 1 \end{bmatrix}$$

computed by the new algorithm (for ease of presentation we ignore the fact that 2-by-2 matrices are handled specially by the algorithm; this same phenomenon holds for larger examples as well). Suppose that $\hat{u}_i$ and $\hat{v}_i$ differ by at most ε_1 from the exact unit singular vectors u_i and v_i of a componentwise relative perturbation $A+\delta A$ of A, where $|\delta A| \leqq \varepsilon_2|A|$. Then if $\varepsilon_2 = o(\varepsilon^{2/3})$, $\varepsilon_1 \cdot \varepsilon_2 = \Omega(\varepsilon)$. In other words, ε_1 is $\Omega(\varepsilon^{1/3})$. Therefore, any attempt to prove a small componentwise relative backward error $o(\varepsilon^{2/3})$ must permit errors in the computed vectors at least as large as $\varepsilon^{1/3} \gg \varepsilon$.

The proof goes as follows. Applying the new algorithm to A (and ignoring the fact that 2-by-2 matrices are handled specially), we set $A_{1,2}$ to 0 and get the columns of the identity matrix as left and right singular vectors. Now we make relative perturbations of size at most $\varepsilon_2 = \varepsilon^\alpha (\alpha > \frac{2}{3})$ in each entry of A (here, $|\varepsilon_{2i}| \leqq \varepsilon_2$):

$$B = A + \delta A = \begin{bmatrix} (1+\varepsilon)(1+\varepsilon_{21}) & \varepsilon(1+\varepsilon_{22}) \\ 0 & 1+\varepsilon_{23} \end{bmatrix}.$$

Compute

$$\begin{aligned} B^TB &= I + \begin{bmatrix} 2\varepsilon + 2\varepsilon_{21} & \varepsilon \\ \varepsilon & 2\varepsilon_{23} \end{bmatrix} + O(\varepsilon^2 + \varepsilon_2^2) \\ &\equiv I + \varepsilon \begin{bmatrix} x & 1 \\ 1 & y \end{bmatrix} + O(\varepsilon^2 + \varepsilon_2^2) \\ &\equiv I + \varepsilon C + O(\varepsilon^2 + \varepsilon_2^2), \end{aligned}$$

where $|x| \leqq 2 + 2\varepsilon_2/\varepsilon$ and $|y| \leqq 2\varepsilon_2/\varepsilon$. We consider the eigenproblem for C. Suppose $[1\ \eta]^T$ is an eigenvector of C; we will show

$$\frac{1}{3 + 4\varepsilon_2/\varepsilon} \leqq |\eta| \leqq 3 + 4\varepsilon_2/\varepsilon,$$

implying the angle between an eigenvector of C and $[1\ 0]^T$ is $\Omega(\varepsilon/\varepsilon_2) = \Omega(\varepsilon^{1-\alpha})$. We compute as follows. If $[1\ \eta]^T$ is an eigenvector of C, η must satisfy $\eta^2 + (x-y)\eta - 1 = 0$ or

$$\eta = \frac{(y-x)}{2} \pm \left(\left(\frac{(y-x)}{2} \right)^2 + 1 \right)^{1/2}.$$

Since $|(y-x)/2| \leqq 1 + 2\varepsilon_2/\varepsilon$, it is easy to see both $|\eta|$ and $|\eta^{-1}|$ are bounded by

$$1 + 2\varepsilon_2/\varepsilon + ((1 + 2\varepsilon_2/\varepsilon)^2 + 1)^{1/2} \leqq 3 + 4\varepsilon_2/\varepsilon$$

as desired.

Now consider the so far ignored perturbation $O(\varepsilon^2 + \varepsilon_2^2)$. The gap between the eigenvalues of C is computed to be $((x-y)^2 + 4)^{1/2} \geqq 2$. Thus the perturbation $O(\varepsilon^2 + \varepsilon_2^2)$ can change the eigenvectors by at most $O(\varepsilon + \varepsilon_2^2/\varepsilon)$. When $\varepsilon_2 = \varepsilon^\alpha$, this is a perturbation of at most $O(\varepsilon^{2\alpha - 1})$. But when $\alpha > \frac{2}{3}$, $2\alpha - 1 > 1 - \alpha$ and so the perturbation cannot change the lower bound $\Omega(\varepsilon^{1-\alpha})$ on $|\eta|$.

Thus a relative perturbation of size ε^α ($\alpha > \frac{2}{3}$) to A means the right singular vectors are least $\Omega(\varepsilon^{1-\alpha}) \equiv \Omega(\varepsilon_1)$ away from the computed right singular vectors. Thus $\varepsilon_1 \cdot \varepsilon_2 = \Omega(\varepsilon)$ as desired.

Since our algorithm handles 2-by-2 matrices as special cases, a 4-by-4 matrix such as

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & \varepsilon & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

could be used in the proof, but the computations are more complicated.

As stated above, rigorous error bounds on the computed singular vectors depend on the singular value distribution, and that the closer together singular values are, the less accurately their corresponding singular vectors can be computed. The "gap" theorem [15, p. 222] expresses this dependency by bounding the angular error in the computed singular vectors by the residual $\|B\hat{v}_i - \hat{\sigma}_i\hat{u}_i,\ B^T\hat{u}_i - \hat{\sigma}_i\hat{v}_i\|$ (the norm of the n by 2 matrix $[B\hat{v}_i - \hat{\sigma}_i\hat{u}_i,\ B^T\hat{u}_i - \hat{\sigma}_i\hat{v}_i]$) divided by the gap (here $\hat{u}_i$, $\hat{v}_i$ and $\hat{\sigma}_i$ are the computed singular vectors and singular value). Standard backward error analysis shows that the residual may be bounded by largest roundoff error committed (which is

$p(n)\varepsilon\|B\|$, $p(n)$ a modest function of n and ε the machine precision). This yields the error bound

$$(9.1)\qquad \max(\theta(\hat{u}_i, u_i), \theta(\hat{v}_i, v_i)) \leqq p(n)\varepsilon\|B\|/gap \equiv p(n)\varepsilon\|B\|/\min|\sigma_i - \sigma_{i\pm1}|;$$

here u_i and v_i are the exact singular vectors.

The bound (9.1) is true for the standard SVD of a dense matrix as well as the new algorithm. A natural question is whether the bound can be improved for the new algorithm applied to the bidiagonal singular value problem. Numerical experience and Proposition 7 in [1] support the following conjecture.

CONJECTURE. Let B be an unreduced bidiagonal matrix with singular values σ_i and left and right singular vectors u_i and v_i. Let the singular vectors computed by the new algorithm be $\hat{u}_i$ and $\hat{v}_i$. Then the errors in $\hat{u}_i$ and $\hat{v}_i$ are bounded by

$$(9.2)\qquad \max(\theta(\hat{u}_i, u_i), \theta(\hat{v}_i, v_i)) \leqq p(n)\varepsilon/\textit{relative gap} \equiv p(n)\varepsilon/\min(|\sigma_i - \sigma_{i+1}|/\sigma_i).$$

The justification for this conjecture is as follows. In § 8 we proved that the zero-shift part of the algorithm is forward stable across a single QR sweep; numerical experience indicates that it is actually forward stable across many QR sweeps. (It is straightforward but tedious to show that after k sweeps, rounding errors can grow by at most a factor which is $O(k)$, but it appears difficult to estimate the constant.) This forward stability means the accumulated transformation matrices are computed accurately. Thus the only serious errors are committed on convergence: setting an off-diagonal to zero. If we use a "conservative" convergence criterion, where only off-diagonals smaller than $\varepsilon \cdot \sigma_{\min}$ are set to zero, the numerator in (9.1) is reduced from $p(n)\varepsilon\|B\|$ to $p(n)\varepsilon\sigma_{\min}$, which implies the conjecture. Extending this argument to the stopping criterion described in § 4 appears difficult, and it is possible that with the more conservative stopping criterion the algorithm will occasionally compute more accurate vectors than the criterion of § 4.

10. Conclusions and open problems. We have described a method for computing all the singular values of a bidiagonal matrix to nearly full machine precision, and showed it to be comparable in speed to the LINPACK SVD algorithm. This computation is justified because small relative errors in the bidiagonal entries (from roundoff in the algorithm or from previous computations) can only cause small relative errors in the singular values, independent of their magnitudes. The technique can be extended to computing the eigenvalues of symmetric positive definite tridiagonal matrices with high relative accuracy as well [1].

A number of open questions remain. First, how accurate are the singular vectors computed by this algorithm? We cannot generally expect high relative accuracy in all singular vectors, because clustered singular values can have arbitrarily ill-conditioned singular vectors. Still, singular vectors might be computable fully accurately as long as the *relative* differences between corresponding singular values and their neighbors are big enough, at least if we use a stopping criterion more conservative than the one in § 4. When in practice is it necessary to compute such accurate singular vectors for tiny clustered singular values? Do applications demand accurate singular vectors, or are tiny residuals sufficient, and if so, how tiny?

Second, since we have shown that it is possible to obtain accurate singular values from accurate bidiagonal matrices, we may ask when it is possible to guarantee accuracy in the reduction to bidiagonal form. This is clearly not possible in general, but for some special classes of matrices (such as positive definite symmetric tridiagonal matrices [1]) reduction to bidiagonal form is accurate. It may also be possible for graded matrices arising from integral equations. For what classes is this true?

Third, how generally can our implicit zero-shift technique be employed to guarantee accurate singular values and eigenvalues? As mentioned in § 3, a similar technique was used in root-free versions of symmetric tridiagonal QR; can it be modified to produce a tridiagonal symmetric QR algorithm that guarantees accurate eigenvalues for at least some interesting classes of symmetric tridiagonal matrices? This question is addressed in [13].

Finally, what is the best parallel algorithm for high accuracy singular values? As mentioned in § 3, zero-shift QR can be parallelized, but it is not as easy to see how to incorporate shifts and convergence testing. In § 6, we showed that bisection and its refinements could be used to compute high accuracy singular values. Such a technique has been successfully parallelized for finding eigenvalues of symmetric tridiagonal matrices [14]. Another possibility is an algorithm based on divide and conquer [11], although it appears difficult to guarantee high accuracy. The answer will probably depend on whether all or just some singular values are desired; in the latter case bisection will likely be superior.

The code is available electronically from James Demmel. It will also be incorporated in the LAPACK linear algebra library [3].

Acknowledgments. The authors acknowledge the suggestions of the referees as well as various colleagues who over the years have pointed out deficiencies of the standard SVD, in particular Augustin Debrulle, Cleve Moler, Beresford Parlett, and G. W. Stewart. This paper was originally presented at Gatlinburg X, Fairfield Glade, Tennessee, in October 1987.

Note added in proof. A complete proof of the conjecture in § 9 may be found in *The Bidiagonal Singular Value Decomposition and Hamiltonian Mechanics*, by Percy Deift, James Demmel, Luen-Chau Li, and Carlos Tomei, Report 458, Computer Science Department, Courant Institute, New York, 1989; SIAM J. Numer. Anal., submitted.

REFERENCES

[1] J. BARLOW AND J. DEMMEL, *Computing accurate eigensystems of scaled diagonally dominant matrices*, Report 421, Computer Science Department, Courant Institute of Mathematical Science, New York University, New York, December 1988; SIAM J. Numer. Anal., to appear.

[2] J. DEMMEL, *Underflow and the reliability of numerical software*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 887–919.

[3] J. DEMMEL, J. DONGARRA, J. DU CROZ, A. GREENBAUM, S. HAMMARLING, AND D. SORENSEN, *Prospectus for the development of a linear algebra library for high performance computers*, Tech. Memorandum No. 97, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, September 1987.

[4] J. DONGARRA AND E. GROSSE, *Distribution of mathematical software via electronic mail*, Comm. ACM, 30 (1987), pp. 403–407.

[5] J. DONGARRA, C. MOLER, J. BUNCH, AND G. W. STEWART, LINPACK *User's Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1979.

[6] F. R. GANTMACHER, *The Theory of Matrices*, Vol. 1, Chelsea, New York, 1959.

[7] G. GOLUB AND W. KAHAN, *Calculating the singular values and pseudo-inverse of a matrix*, SIAM J. Numer. Anal. Ser. B, 2 (1965), pp. 205–224.

[8] G. GOLUB AND C. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.

[9] N. J. HIGHAM, *Efficient algorithms for computing the condition number of a tridiagonal matrix*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 150–165.

[10] IEEE *Standard for binary floating point arithmetic*, ANSI/IEEE Std 754-1985, IEEE Computer Society, Washington, DC, 1985.

[11] E. R. JESSUP AND D. C. SORENSEN, *A parallel algorithm for computing the singular value decomposition of a matrix*, Tech. Memorandom No. 102, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, December 1987.

[12] W. KAHAN, *Accurate eigenvalues of a symmetric tridiagonal matrix*, Tech. Report No. CS41, Computer Science Department, Stanford University, Stanford, CA, July 22, 1966 (revised June 1968).

[13] J. LE AND B. PARLETT, *On the forward instability of the* QR *transformation*, SIAM J. Matrix Anal. Appl., submitted; also Report PAM-419, Center for Pure and Applied Mathematics, University of California, Berkeley, CA, July 1988.

[14] S-S. LO, B. PHILLIPE, AND A. SAMEH, *A multiprocessor algorithm for the symmetric tridiagonal eigenvalue problem*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s155–s165.

[15] B. PARLETT, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[16] G. W. STEWART, *Error and perturbation bounds for subspaces associated with certain eigenvalue problems*, SIAM Rev., 15 (1973), pp. 727–764.

[17] S. VAN HUFFEL, J. VANDEWALLE, AND A. HAEGEMANS, *An efficient and reliable algorithm for computing the singular subspace of a matrix, associated with its smallest singular values*, J. Comput. Appl. Math., 19 (1987), pp. 313–330.

[18] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Oxford University Press, London, 1965.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 5, pp. 913-927, September 1990 006

USE OF THE P^4 AND P^5 ALGORITHMS FOR IN-CORE FACTORIZATION OF SPARSE MATRICES*

M. ARIOLI†§, I. S. DUFF‡, N. I. M. GOULD‡, AND J. K. REID‡

Abstract. Variants of the P^4 algorithm of Hellerman and Rarick and the P^5 algorithm of Erisman, Grimes, Lewis, and Poole, used for generating a bordered block triangular form for the in-core solution of sparse sets of linear equations, are considered.

A particular concern is with maintaining numerical stability. Methods for ensuring stability and the extra cost that they entail are discussed.

Different factorization schemes are also examined. The uses of matrix modification and iterative refinement are considered, and the best variant is compared with an established code for the solution of unsymmetric sparse sets of linear equations. The established code is usually found to be the most effective method.

Key words. sparse matrices, tearing, linear programming, bordered block triangular form, Gaussian elimination, numerical stability

AMS(MOS) subject classifications. 65F50, 65K05, 65F05

1. Introduction. For solving sparse unsymmetric sets of linear equations

$$\mathbf{Ax} = \mathbf{b}, \tag{1.1}$$

Hellerman and Rarick (1971) introduced an algorithm for permuting **A** to bordered block triangular form, which they called the preassigned pivot procedure (P^3). A little later, Hellerman and Rarick (1972) suggested that the matrix should initially be permuted to block triangular form and that the P^3 algorithm should be applied to each diagonal block; they called this the partitioned preassigned pivot procedure (P^4). A potential problem with both of these algorithms is that, when Gaussian elimination is applied to the reordered matrix, some of the pivots may be small or even zero. This leads to numerical instability or breakdown of the algorithm. They intended that small pivots should be avoided, but the published explanation of their algorithm is lacking in detail.

Saunders (1976, p. 222) used column interchanges to avoid this difficulty. Erisman, Grimes, Lewis, and Poole (1985) proposed a cautious variant of P^4 that they called the precautionary partitioned preassigned pivot procedure (P^5). P^5 avoids structurally zero pivots away from the border, but does not address problems associated with small pivots.

Erisman et al. (1985), (1987) performed some extensive numerical tests using as a benchmark the Harwell code MA28 (Duff (1977), Duff and Reid (1979)), which uses the pivotal strategy of Markowitz (1957) and a relative pivot test

$$|a_{kk}^{(k)}| \geqq u \max_{j>k} |a_{kj}^{(k)}| \tag{1.2}$$

* Received by the editors December 28, 1987, accepted for publication (in revised form) July 24, 1989.

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

‡ Computer Science and Systems Division, Harwell Laboratory, Oxfordshire OX11 0RA, United Kingdom. Present address, Central Computing Department, Atlas Centre, Rutherford Appleton Laboratory, Oxfordshire OX11 0QX, United Kingdom.

§ This research was performed while this author was visiting Harwell Laboratory and was funded by a grant from the Italian National Council of Research (CNR), Istituto di Elaborazione dell'Informazione, CNR, via. S. Maria 46, 56100 Pisa, Italy.

on the elements $a_{kj}^{(k)}$ of the kth pivot row. Here u is a preassigned factor, usually set to 0.1. The Erisman et al. (1985) tests showed their P^4 algorithm encountering zero pivots and therefore failing in more than half the test cases. This illustrates that provision for reordering is an essential part of a reliable algorithm. Erisman et al. (1987) found a full 2×2 block that was exactly singular, which illustrates that it is not sufficient to ensure that the diagonal entries are structurally nonzero. They concluded (1987) in favour of the standard Markowitz approach, as represented by MA28.

To make this paper self-contained, we summarize the properties of the reduction to block triangular form in § 2 and of the P^5 and P^4 algorithms in §§ 3 and 4. For detailed descriptions, we refer the reader to Duff, Erisman, and Reid (1986) or Erisman et al. (1985). Both algorithms permute the matrix to a form that is lower triangular with a few "spike columns" projecting into the upper-triangular part. A practical implementation of P^4 or P^5 needs some provision for reordering to avoid small pivots. We consider this in § 5. We have constructed an experimental code to explore these ideas, and the results are presented in § 6. Finally, we present conclusions in § 7.

We assume throughout the paper that there is enough main storage for the computation to be performed without the use of any form of auxiliary storage.

2. Block triangular matrices. Both the P^4 and the P^5 algorithms start by permuting the matrix to block triangular form

$$\begin{pmatrix} \mathbf{A}_{11} & & & & \\ \mathbf{A}_{21} & \mathbf{A}_{22} & & & \\ \mathbf{A}_{31} & \mathbf{A}_{32} & \mathbf{A}_{33} & & \\ \cdot & \cdot & \cdot & \cdot & \\ \mathbf{A}_{N1} & \mathbf{A}_{N2} & \mathbf{A}_{N3} & \cdot & \mathbf{A}_{NN} \end{pmatrix}, \tag{2.1}$$

which allows the system of linear equations

$$\mathbf{Ax} = \mathbf{b} \tag{2.2}$$

to be solved by block forward substitution

$$\mathbf{A}_{ii}\mathbf{x}_i = \mathbf{b}_i - \sum_{j=1}^{i-1} \mathbf{A}_{ij}\mathbf{x}_j, \qquad i = 1, 2, \cdots, N. \tag{2.3}$$

We assume that each block $\mathbf{A}_{ii}$ is irreducible, that is it cannot itself be permuted to block triangular form. There are well-established and successful algorithms for reducing a matrix to this form (see Duff et al. (1986), Chap. 6, for example), and good software is available. It is the treatment of the blocks $\mathbf{A}_{ii}$ that is our concern here.

Because we will subsequently perform a block decomposition of the $\mathbf{A}_{ii}$ blocks, we will use the graph theoretic equivalent term "*strong component*" to identify a block $\mathbf{A}_{ii}$ in the following text.

3. The P^5 algorithm. The P^5 algorithm (Erisman et al. (1985)) first permutes the matrix to block triangular form and then further permutes each strong component to the form illustrated in Fig. 3.1. The general form is of a matrix

$$\begin{pmatrix} \mathbf{B} & \mathbf{C} \\ \mathbf{D} & \mathbf{E} \end{pmatrix}, \tag{3.1}$$

where $\mathbf{B}$ is block lower triangular with full diagonal blocks and each column of $\mathbf{C}$ has a leading block of nonzero entries in the rows of a diagonal block of $\mathbf{B}$ and extends upwards at least as far as the preceding column. We refer to the columns of $\binom{\mathbf{C}}{\mathbf{E}}$ as the

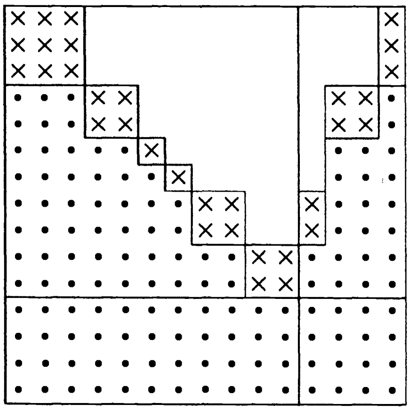


FIG. 3.1. *A strong component after the application of* P^5. *Entries shown as "×" are nonzero and entries shown as "●" may be zero or nonzero.*

"border." For consistency of notation, we refer to any column that projects above the diagonal as a "spike" column, even though they do not look like spikes in the case of the P^5 algorithm. A border column or a column that is not the first of a diagonal block is a spike column, and there are no other spike columns. We refer to the part of a spike column that lies above the diagonal as a "spike."

4. The P^4 algorithm.

The P^4 algorithm leads to the same number of spike columns as the P^5 algorithm, but some of the border columns are moved forward. They still have the desirable property that each spike acts as the border of a bordered block triangular matrix in a properly nested set of such matrices. For example, the matrix of Fig. 3.1 might have the form shown in Fig. 4.1. The two spikes in the middle of the border have moved forward and become separated. At the outer level, the blocks have sizes 3, 6, and 5. The first is a full 3×3 matrix. The last is a bordered form with inner blocks that are full 2×2 matrices. The middle one is a bordered form with inner blocks of sizes 4 and 1, the first of which is a bordered form with blocks of sizes 2 and 1.

Our implementation is as described by Duff et al. (1986). Note, in particular, that this version ensures that the diagonal entries are nonzero unless they are in the border.

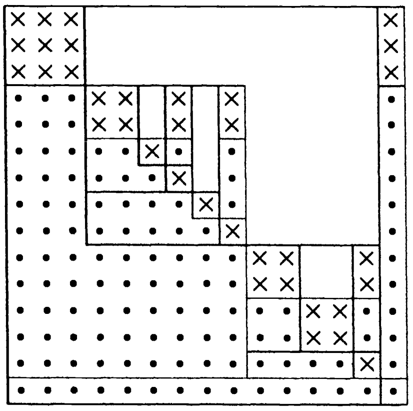


FIG. 4.1. *The matrix of Fig.* 3.1 *after application of* P^4. *Entries shown as "×" are nonzero and entries shown as "●"may be zero or nonzero.*

5. The treatment of spiked matrices. If Gaussian elimination without interchanges is applied to any sparse matrix, including those produced by P^4 and P^5, all fill-in is confined to the spike columns (if column j is not a spike column, it is inactive during steps $1, 2, \cdots, j-1$ of the elimination). Since this produces triangular factorizations of the blocks, we will refer to this as "LU."

An interesting property of the matrices produced by P^4 and P^5 is that they produce a properly nested set; that is, given any pair of spikes, either the set of rows that are cut by the first spike on or above its diagonal is a subset of the corresponding set for the second spike or the two sets of rows do not overlap. Therefore, if Gauss-Jordan elimination is performed by applying row operations to eliminate any entry in the lower triangular part of row 2 (that is, entry (2, 1), if present), then any entry in the upper triangular part of column 2 (that is, entry (1, 2), if present), then all the entries in the lower triangular part of row 3, then all the entries in the upper triangular part of column 3, etc., all fill-in is confined to the spikes (that is, the parts of spike columns that project above the diagonal). Note that if the spikes were not properly nested, fill-in would lengthen some of them. Gauss-Jordan elimination is sometimes called "product form of the inverse" or PFI for short. It is more usual to perform the Gauss-Jordan eliminations column by column.

For numerical stability, Saunders (1976) suggests considering a column interchange whenever the inequality

$$|a_{kk}^{(k)}| \geqq u_1 \max_{i>k} |a_{ik}^{(k)}| \tag{5.1}$$

is not satisfied, where u_1 is a small threshold (often 0.001). He took the largest $a_{kj}^{(k)}, j = k, \cdots, n$, as the pivot, but now recommends (private communication) choosing the first spike column, l, such that

$$|a_{kl}^{(k)}| \geqq u_2 \max_{j \geqq k} |a_{kj}^{(k)}|, \tag{5.2}$$

where u_2 is another threshold (usually 0.1), in order that the structure is corrupted least. Indeed, it is possible for a pivot to fail the test (5.1) and yet pass the test (5.2). Note that a column interchange may corrupt the property of the previous paragraph and hence lead to fill-ins that lengthen later spikes.

In both the P^4 and P^5 algorithms, even after column interchanges, we have the block form

$$\begin{pmatrix} \mathbf{B} & \mathbf{C} \\ \mathbf{D} & \mathbf{E} \end{pmatrix}, \tag{5.3}$$

where **B** and **E** are square and the second block column contains all the border columns. We assume that implicit factorization of this block form is used (see George (1974); see also Duff et al. (1986), p. 61); that is, **B** is factorized and the Schur complement $\mathbf{E} - \mathbf{D}\mathbf{B}^{-1}\mathbf{C}$ is formed as a full matrix and factorized using conventional interchanges, but **C** and **D** are stored in their original form without fill-ins. The Schur complement is formed naturally when LU or Gauss-Jordan factorization is used and there is no need to calculate $\mathbf{B}^{-1}$ explicitly, but note that the number of operations for forming it by the two methods is usually different. Where **B** has a block structure, we could also use an implicit factorization for **B** so that no fill-in is held in its off-diagonal blocks, but have not done this in our experiments because we found that the inner borders were too small to justify the extra complication.

6. Numerical experiments. For our numerical experiments, we have taken the matrices studied by Erisman et al. (1985) and included a few more. We have applied the Harwell code MA28, which performs Gaussian elimination with the ordering of Markowitz (1957) and threshold pivoting. We have used the usual value, $u = 0.1$, for the threshold. MA28 transforms the matrices to block triangular form so that fill-in is confined to the strong components, but implicit factorization of the strong components (see end of § 5) is not available for MA28. We passed the strong components to the Harwell code MC33 that has options for P^4 and P^5 ordering. We then applied the algorithms of § 5 with (u_1, u_2) values of $(0.1, 0.1)$, $(0.001, 0.1)$, and $(0.0, 0.0)$.

The first comment that is worth making concerning our experience is that the block triangular form usually has few nontrivial strong components. Many matrices that occur in practice are irreducible (have only one strong component) and those that are reducible usually have many trivial strong components. The chemical engineering problems considered by Erisman et al. are all reducible and we show the sizes of their strong components in Table 6.1. We also show in Table 6.2 the sizes of the strong components for the linear programming bases BP0, · · · , BP1600 in the Harwell set of sparse matrices. We also use a set of matrices from François Cachard of Grenoble. These matrices arose in the simulation of computing systems (Cachard (1981)) and are all irreducible. The matrices in the test set have some entries that are numerically zero. We processed this data to remove zeros so that the number of entries is less than indicated in the set distributed by Duff, Grimes, and Lewis (1987). In most instances, the runs on the BP matrices showed the same trend as the other sets and did not add to our understanding of the algorithms. We have therefore omitted tables for most of the BP runs. We have also run the remaining problems referenced in the papers of Erisman et al. and have found the relative performance similar to the results displayed.

When the P^4 and P^5 algorithms are applied to the strong components, the final border plays a very important role. Most of the spikes lie within it, as Tables 6.3-6.5 and Figs. 6.1-6.2 show. For the P^5 algorithm, this implies that most of the diagonal blocks are of size 1. We have observed that those not of size 1 are usually of size 2.

TABLE 6.1
The sizes of the strong components of Westerberg's matrices.

Order	No. of components of size 1 or 2	Other sizes
67	1	66
132	55	77
156	133	23
167	90	77
381	5	375
479	165	308
497	291	92; 57; 57
655	197	452
989	269	720
1,505	403	1,099; 3
2,021	521	1,500

TABLE 6.2
The strong component sizes for BP0, · · · , BP1600. *All the matrices have order* 822.

Identifier	No. of components of size 1 or 2	No. of components of size 3 to 9	Other sizes
BP0	822	0	None
BP200	658	12	32, 39, 40
BP400	575	14	22, 161
BP600	523	11	12, 18, 216
BP800	475	15	33, 244
BP1000	446	13	33, 56, 19, 207
BP1200	426	17	33, 65, 220
BP1400	390	12	372
BP1600	431	16	32, 69, 217

TABLE 6.3
The numbers of spikes and border sizes for Westerberg's matrices.

Order	Component size	No. of spikes	Border size, P^5	Border size, P^4
67	66	14	13	11
132	77	6	4	3
156	23	4	4	3
167	77	6	4	3
381	375	75	53	52
479	308	61	42	38
497	92	19	18	16
497	57	1	1	1
497	57	1	1	1
655	452	93	66	54
989	720	98	84	77
1,505	1,099	148	127	116
2,021	1,500	205	175	160

TABLE 6.4
The numbers of spikes and border sizes for the Grenoble set of irreducible matrices.

Order	No. of spikes	Border size, P^5	Border size, P^4
115	19	15	15
185	28	28	28
216	25	25	24
216	25	25	24
343	52	52	42
512	55	55	50
1,107	283	113	100

TABLE 6.5
The numbers of spikes and border sizes for strong components of sizes 10 *or more from the* BP *matrices.*

Identifier	Component size	No. of spikes	Border size, P^5	Border size, P^4
BP200	32	4	2	2
	39	4	3	2
	40	7	3	3
BP400	22	5	2	2
	161	30	21	17
BP600	12	4	3	1
	18	4	3	2
	216	42	27	21
BP800	33	9	6	4
	244	49	38	30
BP1000	33	9	7	5
	56	13	10	6
	19	2	2	2
	207	51	32	26
BP1200	33	9	5	4
	65	15	14	8
	220	49	34	25
BP1400	372	83	54	45
BP1600	32	9	5	4
	69	20	16	8
	217	44	29	24

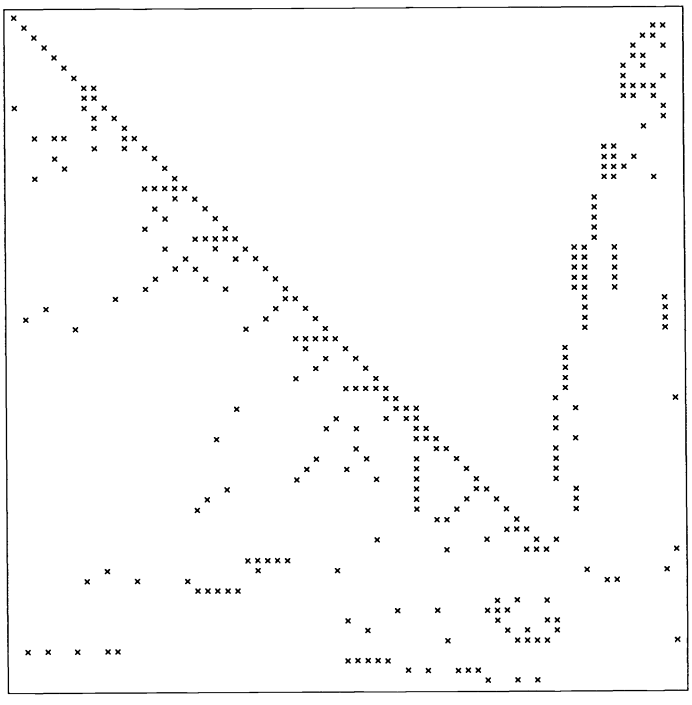


FIG. 6.1. *The matrix of order* 67 *from the Westerberg set after application of* P^5.

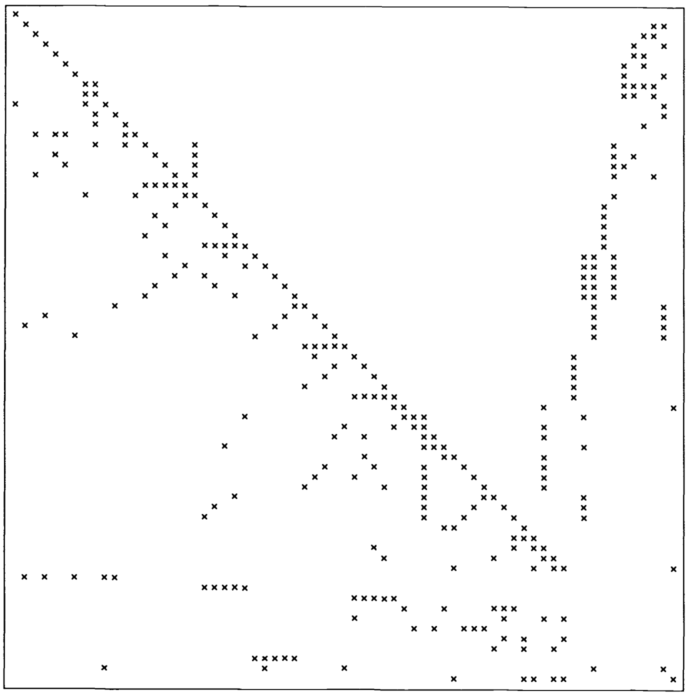


FIG. 6.2. *The matrix of Fig.* 6.1 *after application of* P^4.

In all the tables describing the P^4 and P^5 experiments, we indicate by flop count the number of multiplications, divisions, and additions performed, and we count only the fill-in blocks **B** and **E** of (5.3) since we are using the implicit form of the factorization.

Our experience with Gauss-Jordan elimination, as described in § 5, is that it does indeed often involve less fill-in, but the additional elimination steps lead to a very substantial increase in operation counts; see Tables 6.6 and 6.7. So substantial is this increase that we do not consider this variant further.

TABLE 6.6
Comparison between Gauss-Jordan (GJ) *and* LU *on Westerberg's matrices, using the* P^4 *algorithm and thresholds* $u_1 = 0.1$ *and* $u_2 = 0.1$.

		Fill-in		Factorization flop count (thousands)	
Order	Nonzeros	GJ	LU	GJ	LU
67	294	53	113	7.4	3.2
132	413	5	17	1.2	0.9
156	362	8	14	0.4	0.2
167	506	13	20	1.3	0.9
381	2,134	1,332	2,708	834	137
479	1,888	704	1,307	336	44
497	1,721	129	254	16	8.0
655	2,808	1,447	2,704	931	119
89	3,518	2,889	4,098	2,345	171
1,505	5,414	6,177	7,474	6,060	368
2,021	7,310	13,225	19,506	23,714	1,756

TABLE 6.7
Comparison between Gauss-Jordan (GJ) *and* LU *on Grenoble matrices, using the* P^4 *algorithm and thresholds* $u_1 = 0.1$ *and* $u_2 = 0.1$.

		Fill-in		Factorization flop count (thousands)	
Order	Nonzeros	GJ	LU	GJ	LU
115	421	153	373	43	13
185	975	388	1,085	239	77
216	812	299	571	121	25
216	812	339	3,000	320	127
343	1,310	935	1,853	569	89
512	1,976	1,278	2,515	1,235	161
1,107	5,664	6,731	11,382	14,849	1,744

To discover whether our code is sensitive to the way that the choice is made when the heuristics of the algorithms say that more than one column is equally good (that is, sensitivity to tie-breaking), we ran several problems with their columns randomly permuted. We found little sensitivity. For example, 10 runs of the Grenoble case of order 512 had P^5 borders varying between 51 and 55 and fill-in with LU factorization ($u_1 = u_2 = 0.1$) varying between 2,594 and 3,008.

It can be seen from Tables 6.3 and 6.4 that the P^4 algorithm leads to relatively few spikes being moved forward from the border. We might therefore expect that it would not make a large difference to the fill-in or operation count. This is confirmed in Tables 6.8 and 6.9. Usually, but not always, moving spikes forward leads to an improvement. We therefore prefer the P^4 algorithm.

We have stressed the need for interchanges to avoid small pivots. This is essential if accurate solutions are to be obtained, as is illustrated in Tables 6.10 and 6.11, where the relative residuals are

$$\frac{|r_i|}{|b_i| + \sum_j |a_{ij}|\,|\tilde{x}_j|}, \quad i = 1, 2, \cdots \tag{6.1}$$

TABLE 6.8
Comparison between P^5 *and* P^4 *on Westerberg's matrices, using* LU *factorization and thresholds* $u_1 = 0.1$ *and* $u_2 = 0.1$.

		Fill-in		Factorization flop count (thousands)	
Order	Nonzeros	P^5	P^4	P^5	P^4
67	294	125	113	3.1	3.2
132	413	21	17	0.9	0.9
156	362	15	14	0.2	0.2
167	506	17	20	0.9	0.9
381	2,134	2,712	2,708	139	137
479	1,888	1,576	1,307	53	44
497	1,721	301	254	8.2	8.0
655	2,808	3,776	2,704	177	119
989	3,518	4,317	4,098	199	171
1,505	5,414	11,761	7,474	750	368
2,021	7,310	22,395	19,506	2,064	1,756

TABLE 6.9
Comparison between P^5 *and* P^4 *on Grenoble matrices, using* LU *factorization and thresholds* $u_1 = 0.1$ *and* $u_2 = 0.1$.

		Fill-in		Factorization flop count (thousands)	
Order	Nonzeros	P^5	P^4	P^5	P^4
115	421	373	373	13	13
185	975	1,085	1,085	77	77
216	812	619	571	27	25
216	812	3,014	3,000	132	127
343	1,310	2,657	1,853	132	89
512	1,976	2,996	2,515	193	161
1,107	5,664	13,848	11,382	2,151	1,744

TABLE 6.10
Comparison between $u_1 = 0.0$, 0.001, *and* 0.1 *on Westerberg's matrices, using* P^4 *and* LU *factorization with* $u_2 = 0.1$. *The arithmetic is* IBM *double precision.*

		Max. relative residual			No. of col. interchanges		
Order	Nonzeros	$u_1 = 0.0$	$u_1 = 0.001$	$u_1 = 0.1$	$u_1 = 0.0$	$u_1 = 0.001$	$u_1 = 0.1$
67	294	5×10^{-15}	5×10^{-15}	6×10^{-15}	0	0	0
132	413	3×10^{-9}	1×10^{-11}	2×10^{-15}	0	2	2
156	362	4×10^{-15}	9×10^{-16}	9×10^{-16}	0	1	1
167	506	3×10^{-9}	2×10^{-13}	4×10^{-15}	0	2	3
381	2,134	fails	4×10^{-10}	1×10^{-12}	0	2	6
479	1,888	1×10^{-6}	4×10^{-11}	4×10^{-14}	0	6	10
497	1,721	fails	4×10^{-11}	4×10^{-14}	0	0	3
655	2,808	3×10^{-7}	6×10^{-11}	1×10^{-13}	0	6	10
989	3,518	4×10^{-7}	7×10^{-12}	4×10^{-14}	0	28	39
1,505	5,414	1×10^{-7}	3×10^{-10}	1×10^{-13}	0	42	63
2,021	7,310	4×10^{-6}	1×10^{-9}	2×10^{-13}	0	56	77

TABLE 6.11
Comparison between $u_1 = 0.0$, 0.001, *and* 0.1 *on Grenoble matrices, using* P^4 *and* LU *factorization with* $u_2 = 0.1$. *The arithmetic is* IBM *double precision.*

		Max. relative residual			No. of col. interchanges		
Order	Nonzeros	$u_1 = 0.0$	$u_1 = 0.001$	$u_1 = 0.1$	$u_1 = 0.0$	$u_1 = 0.001$	$u_1 = 0.1$
115	421	5×10^{-10}	5×10^{-10}	1×10^{-13}	0	0	11
185	975	2×10^{-8}	2×10^{-8}	2×10^{-12}	0	0	16
216	812	1×10^{-9}	1×10^{-9}	1×10^{-9}	0	0	0
216	812	fails	fails	4×10^{-14}	0	0	85
343	1,310	1×10^{-9}	1×10^{-9}	1×10^{-9}	0	0	0
512	1,976	4×10^{-2}	4×10^{-2}	4×10^{-2}	0	0	0
1,107	5,664	fails	fails	fails	0	3	24

and $\tilde{x}$ is the computed solution. The number of interchanges is shown in the tables. Interchanges usually lead to an increase in fill-in and operation count (illustrated in Tables 6.12 and 6.13) principally because of exchanges between the border and nonborder columns.

For the Westerberg matrices, the choice of $u_1 = 0.1$ gives good residuals, without an excessive increase in fill-in or operation count; the choice $u_1 = 0.001$ yields higher residuals, but they are reasonably satisfactory.

For the Grenoble matrices, the choice $u_1 = 0.001$ yields unsatisfactory residuals in cases 4, 6, and 7, and the choice $u_1 = 0.1$ yields unsatisfactory residuals in cases 6 and 7. We regard the use of the small value of 0.001 as "living dangerously" and are not surprised by an occasional poor result, but the last two results with the value 0.1 prompted us to investigate further. We found that they were indeed caused by large growth in the size of the matrix entries. For case 6 (order 512), we found that increasing u_1 to 0.2 did not help but that increasing it to 0.5 reduced the maximum relative residual to 1×10^{-6}. For case 7 (order 1107), the values 0.2 and 0.5 reduced the residuals to 1×10^{-9} and 6×10^{-13}, respectively. Increasing u_2 made little difference to the results.

TABLE 6.12
Comparison between $u_1 = 0.0$, 0.001, and 0.1 on Westerberg's matrices, using P^4 *and* LU *factorization with $u_2 = 0.1$.*

		Fill-in			Factorization flop count (thousands)		
Order	Nonzeros	$u_1 = 0.0$	$u_1 = 0.001$	$u_1 = 0.1$	$u_1 = 0.0$	$u_1 = 0.001$	$u_1 = 0.1$
67	294	117	117	113	3.2	3.2	3.2
132	413	18	30	17	1.0	1.1	0.9
156	362	13	14	14	0.2	0.2	0.2
167	506	16	22	20	0.9	1.0	0.9
381	2,134	2,626	2,707	2,708	134	148	137
479	1,888	1,299	1,438	1,307	41	50	44
497	1,721	255	256	254	9.3	9.6	8.0
655	2,808	2,421	2,918	2,704	93	131	119
989	3,518	3,232	2,938	4,098	112	98	171
1,505	5,414	5,834	6,352	7,474	263	260	368
2,021	7,310	16,501	16,353	19,506	1,263	1,178	1,756

TABLE 6.13
Comparison between $u_1 = 0.0$, 0.001, and 0.1 on Grenoble matrices, using P^4 *and* LU *factorization with $u_2 = 0.1$.*

		Fill-in			Factorization flop count (thousands)		
Order	Nonzeros	$u_1 = 0.0$	$u_1 = 0.001$	$u_1 = 0.1$	$u_1 = 0.0$	$u_1 = 0.001$	$u_1 = 0.1$
115	421	227	227	373	6.9	6.9	13
185	975	771	771	1,085	42	42	77
216	812	571	571	571	25	25	25
216	812	584	584	3,000	25	25	127
343	1,310	1,853	1,853	1,853	89	89	89
512	1,976	2,515	2,515	2,515	161	161	161
1,107	5,664	10,281	10,784	11,382	1,113	1,265	1,744

These results illustrate the potential pitfalls with any particular choice of u_1, but our general recommendation is nevertheless for the value 0.1, which we use for the later comparisons in this paper. Note that iterative refinement may be used to improve any solution; in particular, we found it to be successful for the poor factorizations discussed in the previous paragraph.

We have also worked with the strategy of performing a column interchange whenever the inequality (1.2) is not satisfied, which has software advantages if storage by rows is in use. This strategy is equivalent to the use of a value of u_1 that is greater than 1.0. We found that it almost always leads to more column interchanges and hence to more fill-in and work, though its stability properties are better (for instance, we did not have such serious difficulties with the last two Grenoble matrices).

As an alternative to performing column interchanges, we also considered changing the diagonal elements $a_{kk}^{(k)}$ whenever they do not satisfy the pivot test (1.2). The solution of (1.1) may then be obtained using the modification method (see, for example, Duff et al. (1986), pp. 244-247). This technique has the advantage of allowing predefined storage structures but when r diagonal elements are altered it has the disadvantage of requiring the solutions of $r+1$ linear systems, each of whose coefficient matrix is the perturbed matrix. Unfortunately, the size of r is normally about the same as the number of column interchanges that would otherwise have been performed with $u_1=u_2=0.1$, and this makes such an approach prohibitively expensive (see Tables 6.10 and 6.11).

Another alternative is to ignore the pivot test (1.2) but to check the absolute value of each entry on the diagonal and to increase it by a value μ if it is less than that value. Usually the Schur complement is full and we can factorize it by an LU decomposition with interchanges that makes the pivots the largest entries of the columns (or rows). This strategy is particularly useful for P^5, because the fill-in can be confined to the Schur complement. Obviously the solution obtained may be poor, but it is possible to improve the error by performing a few steps of iterative refinement. In Tables 6.14 and 6.15, we show results for Westerberg's matrices and those from Grenoble using a value for μ of 1×10^{-8} (approximately the square root of the machine precision).

This approach does not guarantee that iterative refinement converges because the error in the factorization could be too large. For example, with the Grenoble matrix of order 1107 iterative refinement does not converge, because the factorization is

TABLE 6.14

Results on Westerberg's matrices, using LU *factorization with iterative refinement and* P^5, *incrementing the pivot by* 1×10^{-8} *if it is less than* 1×10^{-8}. *The arithmetic is* IBM *double precision.*

Order	Nonzeros	Fill-in	Flop count (thousands) Factorization	Flop count (thousands) Solution	Num. iter. steps	Error
67	294	125	3.1	1.4	0	5×10^{-15}
132	413	15	0.9	0.9	0	8×10^{-10}
156	362	13	0.2	0.6	0	3×10^{-8}
167	506	13	0.9	1.1	0	4×10^{-10}
381	2,134	2,721	139	33	1	2×10^{-12}
479	1,888	1,478	46	16	1	4×10^{-11}
497	1,721	293	9.1	4.4	0	2×10^{-10}
655	2,808	2,756	106	30	1	5×10^{-11}
989	3,518	3,121	104	30	1	2×10^{-10}
1,505	5,414	10,172	598	69	1	3×10^{-10}
2,021	7,310	18,078	1,397	115	1	2×10^{-9}

TABLE 6.15

Results on Grenoble matrices, using LU *factorization with iterative refinement and* P^5, *incrementing the pivot by* 1×10^{-8} *if it is less than* 1×10^{-8}. *The arithmetic is* IBM *double precision* (*note that the fourth matrix is very ill-conditiond*).

			Flop count (thousands)			
Order	Nonzeros	Fill-in	Factorization	Solution	Num. iter. steps	Error
115	421	222	6.9	6.1	1	8×10^{-16}
185	975	771	42	20	1	5×10^{-13}
216	812	617	27	19	1	4×10^{-17}
216	812	617	27	80	7	6×10^{-2}
343	1,310	2,657	132	50	1	2×10^{-16}
512	1,976	2,996	193	265	5	4×10^{-13}
1,107	5,664	12,723	1,458		Diverged	

unstable with entries of size 1×10^{33}. Our view is that this approach does not show sufficient advantages to compensate for the lack of robustness, and we therefore reject it.

Finally, in Tables 6.16-6.18, we show a comparison between the most satisfactory of our algorithms, P^4 with LU factorization and thresholds $u_1 = u_2 = 0.1$, and the Harwell Markowitz code MA28, with threshold $u = 0.1$. The P^4 algorithm often involves less fill-in, but in most cases it requires more operations, sometimes considerably more.

7. Conclusions. We have compared the P^4 and P^5 variants of the Hellerman-Rarick algorithm and found that P^4 is usually better than P^5, but not by much. The special form of Gauss-Jordan elimination that confines fill-in to the spikes themselves usually requires far more operations than LU factorization and does not always lead to less fill-in. We therefore prefer P^4 with LU factorization.

The use of interchanges is essential if a reliable solution is to be obtained, even though the interchanges may lead to an increase in computation.

We also tried to maintain the structure during the factorization by modifying the pivot when it was too small. We examined the possibility of using updating schemes

TABLE 6.16

Comparison between MA28 *and* P^4 *on Westerberg's matrices, using* LU *factorization and thresholds* $u = u_1 = u_2 = 0.1$.

		Fill-in		Factorization flop count (thousands)		Solution flop count (hundreds)	
Order	Nonzeros	MA28	P^4	MA28	P^4	MA28	P^4
67	294	267	113	2.1	3.2	11	8.5
132	413	105	17	0.5	0.9	6.3	5.3
156	362	26	14	0.1	0.2	1.5	1.2
167	506	97	20	0.5	0.9	6.2	5.4
381	2,134	1,941	2,708	25	137	76	102
479	1,888	1,104	1,307	9.8	44	44	54
497	1,721	299	254	2.4	8.0	19	20
655	2,808	2,223	2,704	22	119	80	101
989	3,518	1,194	4,098	7.2	171	69	137
1,505	5,414	1,984	7,474	12	368	109	233
2,021	7,310	2,659	19,506	16	1,756	147	505

TABLE 6.17
Comparison between MA28 *and* P^4 *on Grenoble matrices, using* LU *factorization and thresholds* $u = u_1 = u_2 = 0.1$.

		Fill-in		Factorization flop count (thousands)		Solution flop count (hundreds)	
Order	Nonzeros	MA28	P^4	MA28	P^4	MA28	P^4
115	421	654	373	5.6	13	20	19
185	975	3,134	1,085	63	77	80	53
216	812	2,544	571	39	25	65	29
216	812	2,156	3,000	25	127	57	128
343	1,310	5,334	1,853	119	89	129	67
512	1,976	11,545	2,515	402	161	265	94
1,107	5,664	39,316	11,382	1,928	1,744	889	403

TABLE 6.18
Comparison between MA28 *and* P^4 *on* BP *matrices, using* LU *factorization and thresholds* $u = u_1 = u_2 = 0.1$.

			Fill-in		Factorization flop count (thousands)		Solution flop count (hundreds)	
Identifier	Order	Nonzeros	MA28	P^4	MA28	P^4	MA28	P^4
BP0	822	3,276	0	0	0	0	0	0
BP200	822	3,802	106	25	0.8	1.3	11	10
BP400	822	4,028	266	196	2.0	5.2	20	22
BP600	822	4,172	484	358	3.5	11	30	34
BP800	822	4,534	681	874	5.9	31	41	56
BP1000	822	4,661	766	827	7.1	31	47	63
BP1200	822	4,726	959	843	9.2	37	54	69
BP1400	822	4,790	1,295	1,747	11.4	105	65	91
BP1600	822	4,841	872	734	8.4	34	53	68

but found them to be prohibitively expensive. We also considered iterative refinement and found this sometimes to be very competitive, but disliked its lack of robustness.

Apart from numerical considerations, Erisman et al. (1985), (1987) found the P^5 algorithm to be competitive with the Markowitz algorithm, but were pessimistic about being able to ensure numerical stability. Our comparisons, using the same threshold factor in the two algorithms and taking the number of operations into account, confirm that the Markowitz algorithm is comparable with respect to fill-in and indicate that it is usually superior with respect to factorization operation count. Of course, it should be borne in mind that the Hellerman–Rarick algorithms never need access by rows and are therefore better suited to out-of-core working. They also have a less expensive analysis phase. For example, on the Grenoble matrix of order 1107, our analysis time was 1.2 seconds, whereas MA28 took 60 seconds for the phase that performs both analysis and factorization.

Acknowledgments. We express our thanks to both referees, whose comments have substantially improved the paper. We are also indebted to Michael Saunders for drawing our attention to the double-threshold strategy for column interchanges.

REFERENCES

F. CACHARD (1981), *Logiciel numerique associé à une modelisation de systèmes informatiques*, Thèse, Université Scientifique et Médicale de Grenoble, et l'Institut National Polytechnique de Grenoble, Grenoble, France.

I. S. DUFF (1977), MA28—*A set of Fortran subroutines for sparse unsymmetric linear equations*, Report AERE R8730, Her Majesty's Stationery Office, London, United Kingdom.

I. S. DUFF AND J. K. REID (1979), *Some design features of a sparse matrix code*, ACM Trans. Math. Software, 5, pp. 18-35.

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

I. S. DUFF, R. G. GRIMES, AND J. G. LEWIS (1987), *Sparse matrix test problems*, ACM Trans. Math. Software, 15, pp. 1-14.

A. M. ERISMAN, R. G. GRIMES, J. G. LEWIS, AND W. G. POOLE, JR. (1985), *A structurally stable modification of Hellerman-Rarick's* P^4 *algorithm for reordering unsymmetric sparse matrices*, SIAM J. Numer. Anal., 22, pp. 369-385.

A. M. ERISMAN, R. G. GRIMES, J. G. LEWIS, W. G. POOLE, JR., AND H. D. SIMON (1987), *Evaluation of orderings for unsymmetric sparse matrices*, SIAM J. Sci. Statist. Comput., 8, pp. 600-624.

A. GEORGE (1974), *On block elimination for sparse linear systems*, SIAM J. Numer. Anal., 11, pp. 585-603.

E. HELLERMAN AND D. C. RARICK (1971), *Reinversion with the preassigned pivot procedure*, Math. Programming, 1, pp. 195-216.

——— (1972), *The partitioned preassigned pivot procedure* (P^4), in Sparse Matrices and Their Applications, D. J. Rose and R. A. Willoughby, eds., Plenum Press, New York, pp. 67-76.

H. M. MARKOWITZ (1957), *The elimination form of the inverse and its application to linear programming*, Management Sci., 3, pp. 255-269.

M. A. SAUNDERS (1976), *A fast, stable implementation of the simplex method using Bartels-Golub updating*, in Sparse Matrix Computations, Bunch and Rose, eds., Academic Press, New York, pp. 213-226.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 5, pp. 928-949, September 1990

A MODIFIED SCHWARZ–CHRISTOFFEL TRANSFORMATION FOR ELONGATED REGIONS*

LOUIS H. HOWELL† AND LLOYD N. TREFETHEN‡

Dedicated to the memory of Peter Henrici.

Abstract. The numerical computation of a conformal map from a disk or a half plane onto an elongated region is frequently difficult, or impossible, because of the so-called crowding phenomenon. This paper shows that this problem can often be avoided by using another elongated region, an infinite strip, as the standard domain. A transformation similar to the Schwarz-Christoffel formula maps this strip onto an arbitrary polygonal channel, and a slightly modified transformation maps an elongated rectangle onto an arbitrary closed polygon. By using robust and efficient software for numerical integration and solution of the parameter problem, high-accuracy maps of distorted regions with aspect ratios as high as thousands to one are constructed. The modified mapping method has natural applications in fluid mechanics and electrical engineering.

Key words. Schwarz-Christoffel transformation, conformal mapping, elliptic functions, crowding, grid generation

AMS(MOS) subject classifications. 30C30, 30C20

1. Introduction.

The Schwarz-Christoffel transformation

$$f(z)=A\int^{z}\prod_{j=1}^{n}(z'-z_j)^{\gamma_j}\,dz'+B \tag{1.1}$$

provides an explicit representation of any conformal map of the unit disk or the upper half plane onto any simply connected polygonal region, with or without corners at infinity. There are two well-known computational problems associated with the use of this formula for computing such maps numerically. First, the integral cannot be evaluated analytically except in special cases, and must be approximated by some numerical procedure. Second, while the parameters γ_j are determined by the angles at the vertices w_j of the polygon, the corresponding "prevertices" $z_j=f^{-1}(w_j)$ cannot be determined, in general, a priori and must be obtained iteratively via the solution of a system of nonlinear equations. SCPACK, a robust Fortran package for solving these problems, was provided a few years ago by Trefethen [28], [29] and has been widely used for a variety of applications.[1] Other successful implementations of the Schwarz-Christoffel formula include those of Reppe [26], Davis [3], Floryan [10], Hoekstra [19], and Dias [5].

Standard Schwarz-Christoffel programs fail, however, on some seemingly very simple polygons. They cannot, for example, map a rectangle with an aspect ratio of only 20 to 1, or most other regions with a similar degree of elongation. The reason for this is an intrinsic property of conformal maps that sometimes goes by the name of the "crowding phenomenon" in the literature of numerical conformal mapping (see [7], [14], [18], [21], [31]). Whenever a disk or half plane is mapped to an elongated

* Received by the editors July 5, 1988; accepted for publication (in revised form) June 16, 1989. This work was supported by U.S. Air Force grant AFOSR-87-0102.

† Lawrence Livermore Laboratory, L-419, Livermore, California 94550.

‡ Department of Mathematics, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139.

[1] Available by electronic mail from the "Netlib" facility [6].

region, some of the prevertices are located exponentially close together. For aspect ratios beyond 10 or 20 some groups of prevertices are likely to merge together in finite precision arithmetic, making the evaluation of (1.1) effectively impossible.

Fortunately, many of the distorted regions that come up in applications are highly elongated in only one direction. Indeed, the goal in such problems is often to map the region onto a channel or rectangle for purposes of grid generation or to obtain an exact or simplified solution; a disk or half plane is introduced only as an intermediate step. The purpose of this paper is to show that in such cases, the problem of crowding can be largely eliminated by dispensing with the intermediate domain and mapping directly from an infinite strip, which can be easily transformed to a rectangle if desired. In the language of numerical analysis, constructing the conformal map from a strip to an elongated polygon is often a well-conditioned problem, but conventional algorithms for it are unstable because they depend upon the solution of an ill-conditioned subproblem. Our algorithm is stable because it avoids the subproblem.

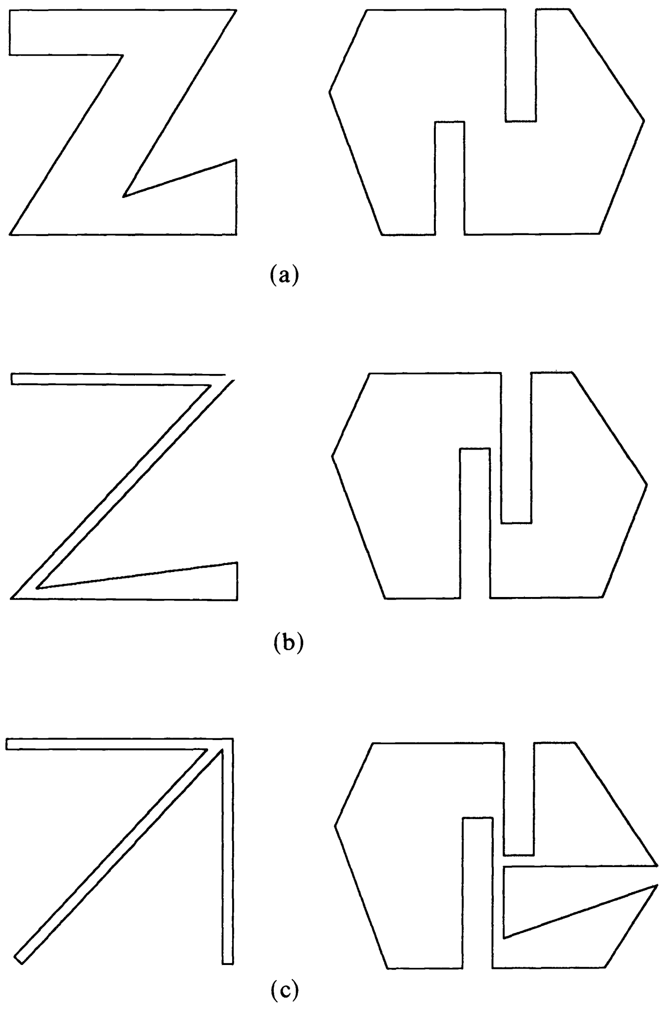


FIG. 1. (a) *Polygons that can be mapped by the standard Schwarz-Christoffel transformation*; (b) *polygons that can be mapped by the strip transformation*; *and* (c) *polygons that cannot be mapped by either method.*

The capabilities described in this paper are summarized in Fig. 1(a)-1(c). Many polygons, those without extreme elongation in any direction, can be mapped by the standard Schwarz-Christoffel methods embodied in SCPACK (Fig. 1(a)). Polygons that are highly elongated in one direction (as well as those that are not elongated) can be mapped by the modified Schwarz-Christoffel methods described here (Fig. 1(b)). For polygons elongated in several directions, different methods will be required (Fig. 1(c)).

The basis of our algorithm is a formula similar to (1.1) for mapping an infinite strip onto an arbitrary polygonal channel (§ 3), and a variation of this formula for mapping a rectangle onto a closed polygon (§ 6). These formulas are not essentially new; the first one dates back at least to Woods [33] and has been used previously by Davis [3], Sridhar and Davis [27], and Floryan [9], [10] for generating computational grids for internal flow problems. The present work differs, however, in emphasizing the crowding phenomenon and in considering the mapping of rectangles to closed polygons as well as other variations. Although it is impossible to be certain in the absence of a direct comparison of computer programs, we believe that our solution method is robust enough to permit the mapping of more complicated regions than those attempted previously. Possible applications of this work include the solution of two-dimensional potential flow problems, the construction of computational grids, the calculation of circuit properties in integrated circuit design [30], and the application of boundary conditions in vortex-method simulations of high Reynolds number flow [15]. For the last example, boundary conditions in vortex calculations, it would be natural to combine the conformal map to an infinite strip discussed here with L. Greengard's recent algorithm for fast calculation of vortex interactions in such a strip [17].

A general method for deriving certain types of Schwarz-Christoffel variations, including the strip formula used here, is presented in [11]. An informal survey of such variations and of applications of Schwarz-Christoffel maps can be found in [32].

Although this paper considers only Schwarz-Christoffel maps, similar ideas might prove useful for more general conformal mapping problems. For example, it would be natural to investigate variants of the Theodorsen, Wegmann, or Hübner methods for mapping an infinite strip onto an elongated region with a curved boundary (see [18], [31]).

2. The crowding problem. The phenomenon of crowding began to be widely recognized as an obstacle to successful numerical conformal mapping around 1980; the term "crowding" itself is due to Menikoff and Zemach [21]. To illustrate this phenomenon, we will examine a simple example that can be treated analytically. The Jacobian elliptic function $\operatorname{sn}(z|m)$ maps the rectangle with corners $-K$, K, $K+iK'$, and $-K+iK'$ to the upper half plane, with the images of the corners being ± 1 and $\pm m^{-1/2}$ (Fig. 2). The constants K and K' are complete elliptic integrals with parameters m and $m_1 = 1-m$, respectively, so only one of K, K', and m can be specified independently. A summary of the properties of elliptic functions and elliptic integrals, including all of the material used in this paper, can be found in [1].

The conformal modulus μ of the rectangle is $K'/2K$, and $m^{1/2}$ can be used as a measure of the crowding effect since it is the ratio of the smallest to the largest length scales in the upper half plane. When μ is large the following asymptotic relationships hold:

$$K \sim \frac{\pi}{2}, \qquad K' \sim \pi\mu, \qquad m^{1/2} \sim 4\, e^{-\pi\mu}. \tag{2.1}$$

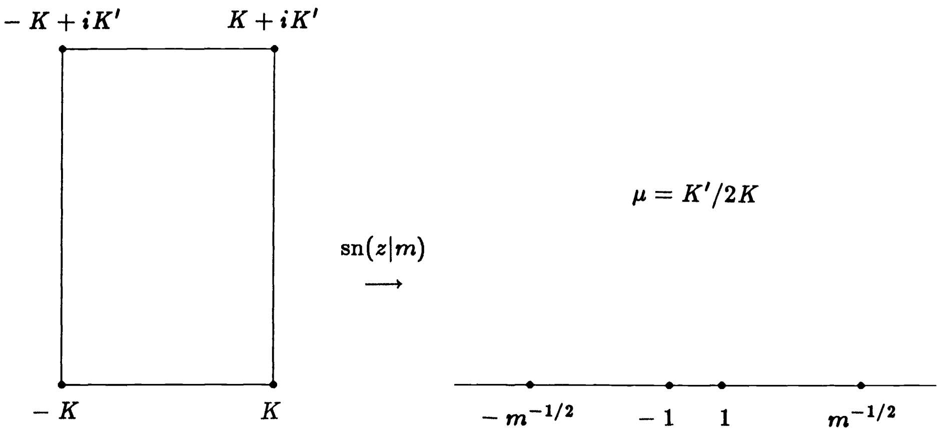


FIG. 2. *Conformal map from a rectangle to the upper half plane.*

Thus for a modulus of only $\mu = 10$, for instance, the important length scales in the upper half plane vary by factors on the order of 10^{13}.

It might seem that these difficulties are surmountable, since in this example all of the crowding occurs near the origin. In floating-point arithmetic the four numbers ± 1 and $\pm m^{-1/2}$ can all be represented to full accuracy, even though the first two may be many orders of magnitude smaller than the last two. There are several reasons, though, why this is an unsatisfactory approach. First, many computers have a range of permissible exponents too restricted to deal with conformal moduli greater than 25 or 50. Second, such a highly distorted mapping would be useless in many applications. Finally, the direct numerical evaluation of the integral (1.1) in such a situation would require a highly specialized quadrature algorithm. A natural first step in such an algorithm would be to change the variable of integration to $\log(z')$, which is in fact equivalent to using the strip transformation.

Figure 3 shows the crowding effect in the conformal map from the unit disk to a mildly elongated region. The four rectangles have moduli 1, 2, 3, and 4, and the internal lines shown are the images of radii to the four prevertices and of equally spaced concentric circles in the unit disk. Due to the conformal nature of each mapping, the innermost circle in each plot is nearly similar to the original disk, so in effect these figures show both the domain and range of each transformation. In particular, we can see the relative positions of the prevertices, and their angular separations are listed in the figure. In the bottom plot, with a conformal modulus of only 4, each of the two pairs of prevertices appears to the eye as a single point. For moduli three or four times larger than this the pairs fuse together in floating-point arithmetic, and the computation by standard methods becomes impossible.

It should be emphasized that crowding occurs when any portion of a domain is elongated. If the rectangles in Fig. 3, for instance, were all extensions of a larger region to the left, then each one would still experience a crowding effect like that shown in the figure. A strongly acute outward-pointing corner can cause a similar problem (for a mild example, see the barb on the arrow in Fig. 10(a)). The methods considered in this paper do not eliminate these secondary crowding effects, which degrade the local accuracy of the mapping and may in extreme cases cause the solution method to fail. In many cases, however, high accuracy can still be obtained in the remainder of the domain.

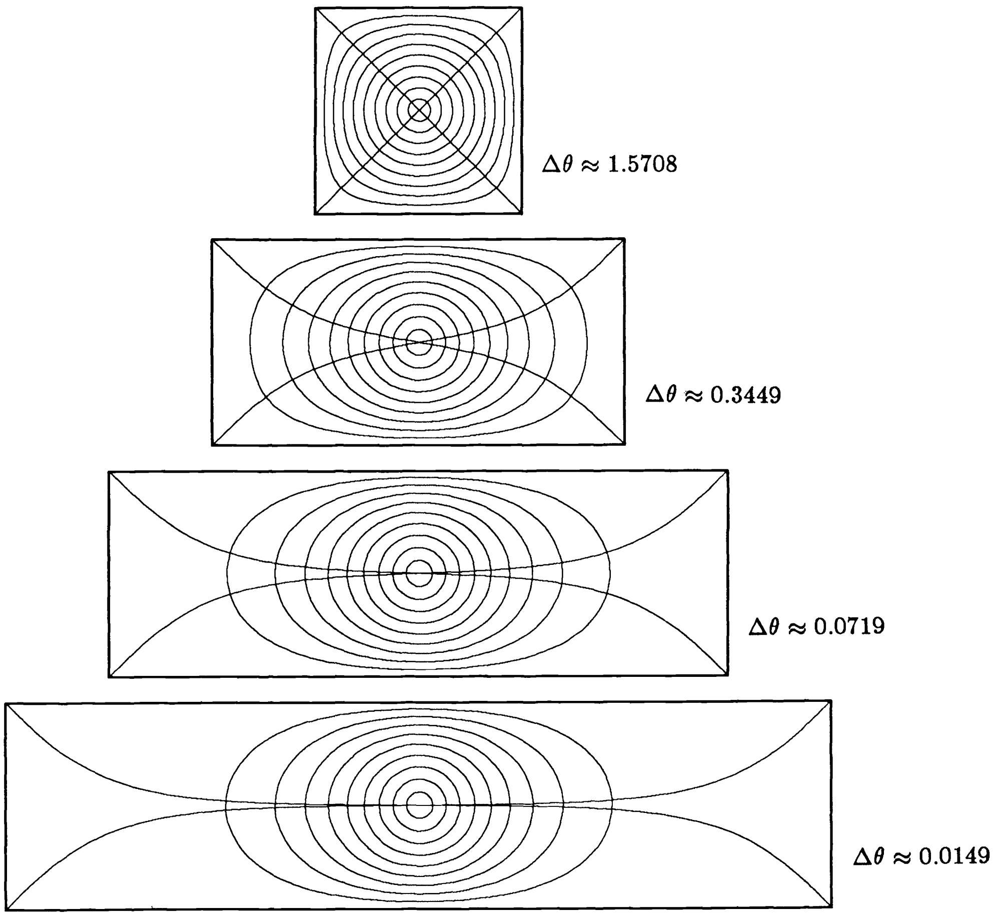

$\Delta\theta \approx 1.5708$
$\Delta\theta \approx 0.3449$
$\Delta\theta \approx 0.0719$
$\Delta\theta \approx 0.0149$

FIG. 3. *Conformal maps of the unit disk onto four rectangles.*

3. The strip transformation. In this section we derive a formula for the conformal map of an infinite strip onto a polygon (3.5); essentially the same formula has been derived earlier (in different ways) by Davis [3] and by Floryan and Zemach [11]. We shall not provide a proof that any conformal map of an infinite strip to a polygon can be represented in this way, but it is true, and a proof can be readily obtained from the standard Schwarz-Christoffel theorem by means of the transformation $e^{\pi z}$ from a strip to a half plane [3].

Figure 4(a)-4(c) defines the geometry of our strip mapping problem: We want to find a conformal map f^* from an infinite strip of width 1 to an infinite polygonal channel P^*. Our notation is that z_j^* and $w_j^* = f^*(z_j^*)$ denote prevertices and vertices for this desired conformal map, while z_j and $w_j = f(z_j)$ correspond to the conformal map onto a polygon $P \approx P^*$ obtained during the course of the numerical solution. The prevertices z_j lie in counterclockwise order around the strip, starting with z_1 on the lower left, proceeding through z_M on the lower right, and ending with z_N on the upper left. The corresponding vertices of the image polygon are denoted by w_j, and the turning angle at w_j is $-\pi\gamma_j$.

Here is the fundamental idea behind the Schwarz-Christoffel map (1.1): the derivative $f'(z)$ has piecewise constant argument on the real axis, which jumps at each prevertex z_j by $-\pi\gamma_j$. To devise a transformation that maps an infinite strip onto an

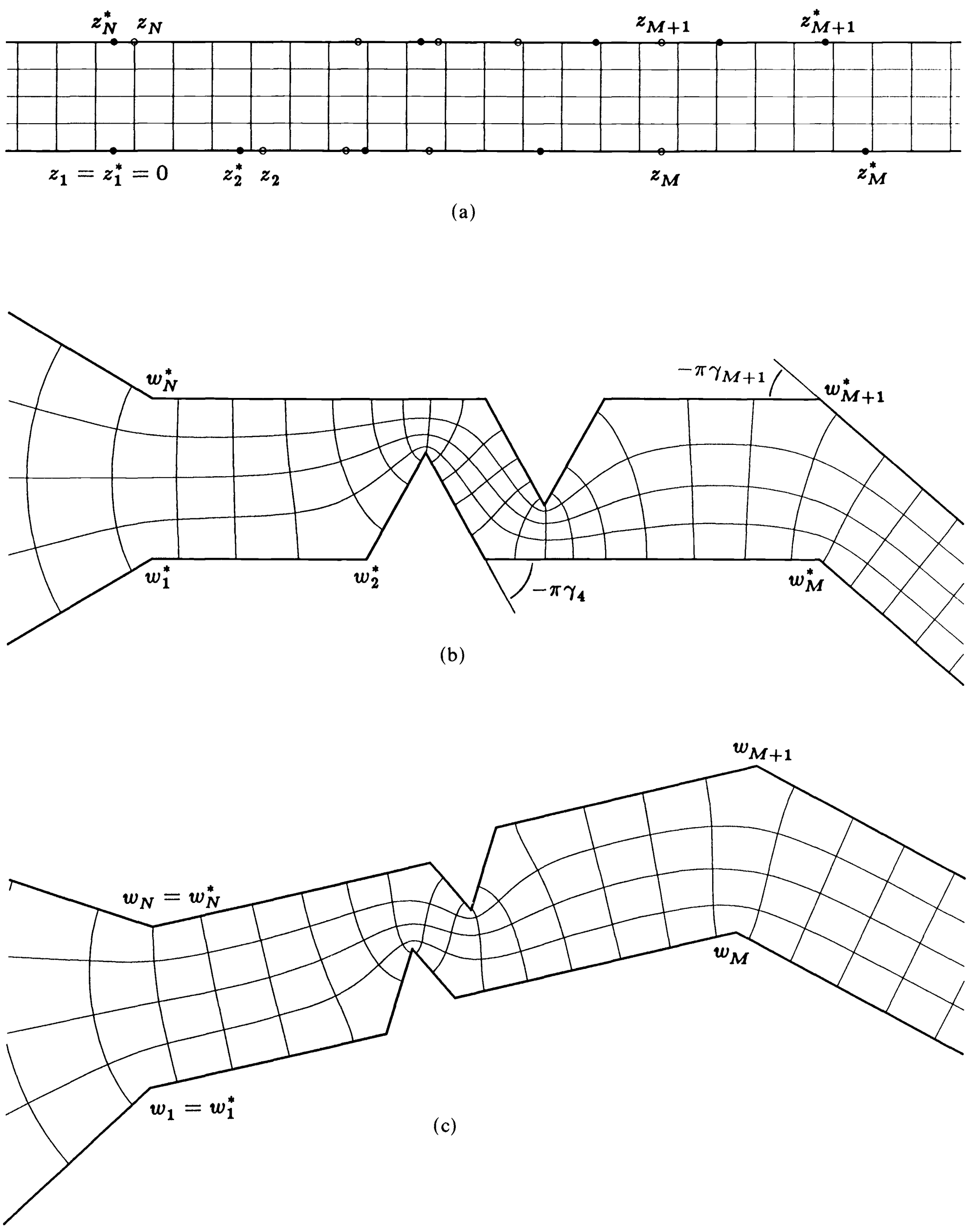


FIG. 4. (a) *Prevertices on the strip: solid dots show the correct values* z_j^*, *and open circles show an incorrect set of values* z_j; (b) *target polygon* P^* *defined by vertices* w_j^*; (c) *polygon* P *defined by incorrect vertices* w_j.

arbitrary polygon, we can utilize the same idea. Specifically, let us derive a function of the form

$$f(z) = A \int^z \prod_{j=1}^{n} f_j(z')\, dz' + B, \tag{3.1}$$

where each factor f_j maps the strip as shown in Fig. 5. The effect of each of these factors is to introduce a corner into one side of the strip while leaving the other side

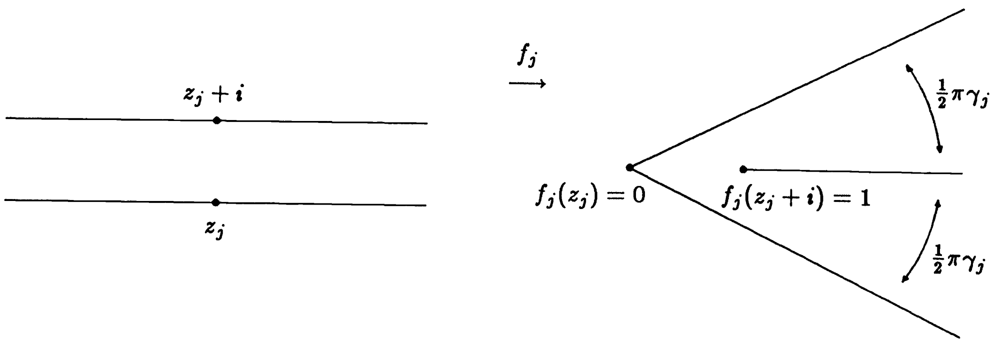


FIG. 5. *A single factor f_j, shown for $\gamma_j > 0$.*

a straight line. The product of several such factors will introduce all of the necessary corners.

The appropriate factors f_j that meet these specifications are very similar to those used in the usual Schwarz-Christoffel formula: For z_j on the lower side of the strip,

$$f_j(z) = \left\{-i \sinh\left[\frac{\pi}{2}(z - z_j)\right]\right\}^{\gamma_j}, \qquad 1 \leqq j \leqq M, \tag{3.2}$$

and for z_j on the upper side,

$$f_j(z) = \left\{-i \sinh\left[-\frac{\pi}{2}(z - z_j)\right]\right\}^{\gamma_j}, \qquad M+1 \leqq j \leqq N. \tag{3.3}$$

In each of these factors the effect of the sinh function is to fold the opposite side of the strip from z_j onto part of the imaginary axis, while the side containing z_j is sent to the real axis. Each section of the boundary is therefore mapped to a line of constant argument, and these remain lines of constant argument after the function is raised to the power γ_j. This property is necessary if the sides of the target domain are to be straight lines. The factors of $-i$ in these equations are mathematically unnecessary, since they can be absorbed into the complex constant A. We have included them, however, to direct the branch cuts of the f_j away from the strip. (Conventionally, and in Fortran, these branch cuts are located on the negative real axis.)

The functions f_j of (3.2) and (3.3) introduce the required angles at the vertices on both sides of the channel, but they always produce equal divergence angles at $\pm\infty$. If we let θ_- and θ_+ be the desired divergence angles at $-\infty$ and $+\infty$, respectively, then the additional factor

$$f_0(z) = \exp[\tfrac{1}{2}(\theta_+ - \theta_-)z] \tag{3.4}$$

provides the necessary adjustment. The full strip transformation is thus given by

$$f(z) = A \int^z \prod_{j=0}^{n} f_j(z')\, dz' + B, \tag{3.5}$$

where the individual functions f_j are defined by (3.1)-(3.4).

4. Solving the parameter problem. How can the prevertices z_j^* be efficiently determined? If prevertices z_j are placed on the correct sides of the strip and in the proper order, but otherwise distributed at random, then the image polygon will in general have the correct angles but incorrect side lengths (Fig. 4). Some kind of iteration must be carried out to find the values z_j^* so that the side lengths come out correct.

As usual in conformal mapping, three real parameters must be specified in order to make f unique. In mapping an infinite strip to an infinite channel, it is natural that the ends of the strip should map to the ends of the channel, so two of these are determined immediately. We have specified the third parameter by fixing z_1^* at the origin; the remaining $N-1$ prevertices $z_2^*, \cdots, z_N^*$ are now the unknowns to be determined iteratively. If the constants A and B in (3.5) are used to fix the positions of w_1 and w_N, then there are correspondingly $N-1$ real geometric conditions—$N-2$ side lengths and one angle—needed to completely specify the shape of the channel.

4.1. Solution via side-length iteration. One popular method for determining the prevertex positions is a simple iterative scheme used by Davis [3] for the standard Schwarz-Christoffel formula. The idea is to make an initial guess for the z_j and then improve it by assuming that the length of each side of the image polygon is roughly proportional to the length of the corresponding interval on the real axis. Thus each interval between prevertices is adjusted according to the formula

$$(4.1) \qquad (z_{j+1}-z_j)_{\text{new}} := (z_{j+1}-z_j) \cdot \frac{|w_{j+1}^* - w_j^*|}{|w_{j+1}-w_j|}.$$

By iterating this procedure it is hoped that the correct solution can be obtained to the desired accuracy. This method has also been used by Floryan [9], [10] and Sridhar and Davis [27], and is quite dependable for many problems. We believe it is not the best choice for a general algorithm, however, for the following reasons:

- When used with the strip transformation, the method gives no information about the position of z_N^*, the leftmost prevertex on the top side of the strip. Sridhar avoids this problem by restricting attention to channels where symmetry implies $z_N^* = i$. Floryan uses a double iteration for the asymmetric case—a one-dimensional secant iteration determines a value for z_N at each step of the global iteration (4.1).
- The proportionality assumption can be violated by difficult problems in at least two different ways. First, it assumes that only the preimages at the endpoints of an interval have a major effect on the length of the corresponding side, and this condition is violated when crowding occurs. Second, if the two singularities at the endpoints are strong, i.e., the interior angles are acute, then the length of the side may actually decrease as the prevertex separation increases. Even with the standard Schwarz-Christoffel formula there are geometries for which (4.1) fails to converge, and with the strip transformation we have the added problem that singularities on the opposite side of the strip may also strongly influence an interval.

  Despite these difficulties, the iteration (4.1) often converges within at most a few tens of iterations, particularly on relatively straightforward problems like those shown in Figs. 4 and 6. On a region like that shown in Fig. 8, however, it diverges even when started very near the solution. (Figure 8 actually shows an example of a map from a rectangle to a closed polygon. The same geometry can be treated as a channel, though, if the right angles at the ends are replaced by straight angles. The prevertex z_N is so far "upstream" in this case that it does not cause significant problems; the main area of difficulty for (4.1) is at the other end of the figure. On this example we have used the high-accuracy adaptive quadrature methods described in § 5—though somewhat inefficient, these routines give quite reliable error bounds, so the failure cannot simply be caused by an inaccurate quadrature algorithm.)

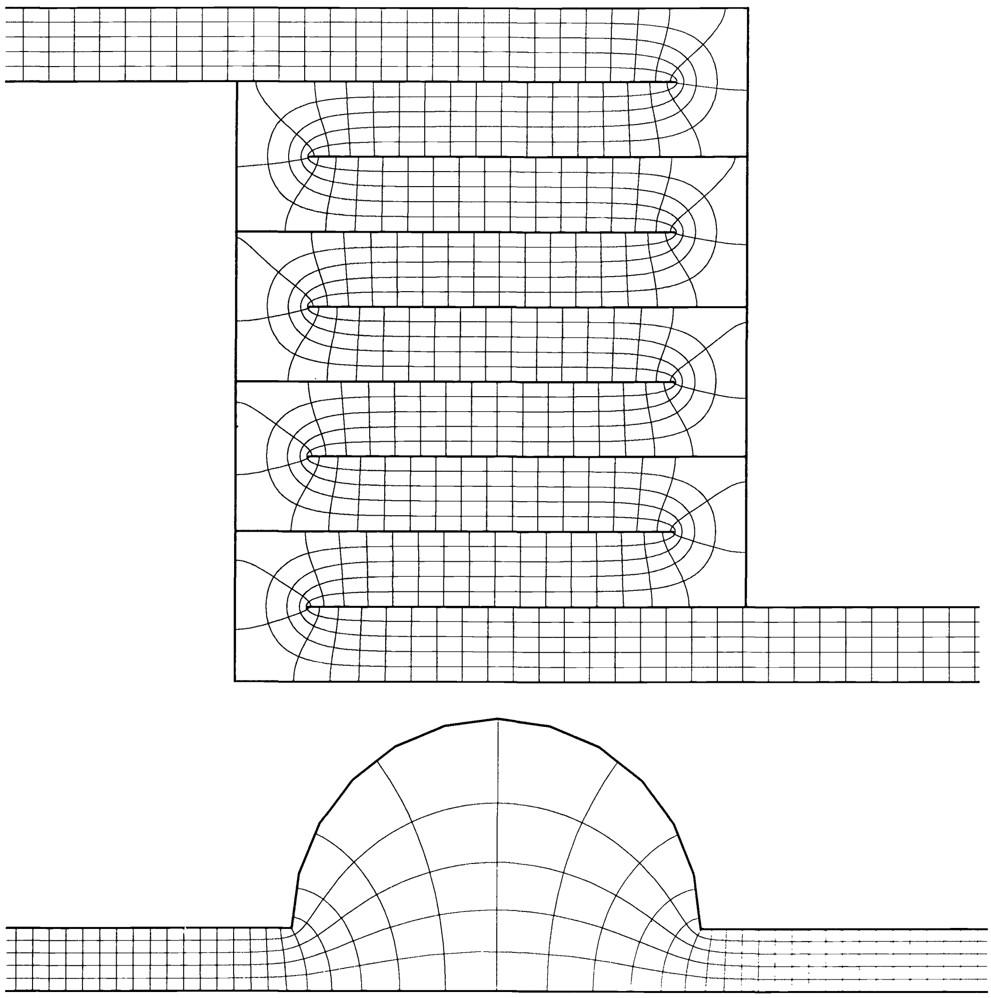


FIG. 6. *Two channels with large N.*

Further examples supporting these claims will be presented in a future paper.

- The convergence is only linear, which may be a disadvantage for high-accuracy computations.
- Many Schwarz-Christoffel problems that arise in applications come with additional conditions to be satisfied. For example, the conformal modulus μ might be specified in advance and one of the side lengths left unspecified. In such situations one has a "generalized parameter problem" to solve [30], which may not be an easy matter if one is using an iteration like (4.1) that is dependent on geometric insight.

4.2. Solution via secant iteration. In our own calculations we have instead viewed the parameter problem as a general system of nonlinear equations $\mathbf{F}(\mathbf{x}^*)=\mathbf{0}$ to be solved numerically; this is an old idea. The normalization described above fixes z_1, so an obvious choice for the $N-1$ independent variables might be $\operatorname{Re}(z_2), \cdots, \operatorname{Re}(z_N)$. This choice leads to a constrainted system, however, since the prevertices on each side of the strip must appear in the proper order. To remove the constraints, we have used a change of variables similar to the one in SCPACK:

$$x_j = \begin{cases} \operatorname{Re}(z_N), & j=1, \\ \log(z_j - z_{j-1}), & 2 \leqq j \leqq M, \\ \log(z_j - z_{j+1}), & M+1 \leqq j \leqq N-1. \end{cases} \tag{4.2}$$

As for the dependent variables, we first compute the positions of $w_1, \cdots, w_N$ by integrating (3.5) and using the constants A and B to fix w_1 and w_N at their correct positions. The $N-1$ functions to be set to zero are then given by

$$F_j = \begin{cases} \operatorname{Im}\left[\log\left(\dfrac{w_2 - w_1}{w_2^* - w_1^*}\right)\right], & j=1, \\ \operatorname{Re}\left[\log\left(\dfrac{w_j - w_{j-1}}{w_j^* - w_{j-1}^*}\right)\right], & 2 \leqq j \leqq M, \\ \operatorname{Re}\left[\log\left(\dfrac{w_j - w_{j+1}}{w_j^* - w_{j+1}^*}\right)\right], & M+1 \leqq j \leqq N-1. \end{cases} \tag{4.3}$$

F_1 is an angle, and each of the other F_j involves the logarithm of a side length. The use of logarithms improves the scaling of the problem when some sides are much longer than others, as often occurs with elongated regions.

We have experimented with three nonlinear equations packages for solving this problem: Powell's subroutine NS01A [25], the Minpack routine HYBRD [22], and an implementation of Schnabel's pseudocode from Dennis and Schnabel [4]. All three are based on a hybrid (dogleg) quasi-Newton algorithm with secant updates. On average we obtained slightly better results with HYBRD, and in addition one of our test problems caused Powell's code to fail. (This difficulty was apparently due to an overly strict stopping criterion rather than a fundamental failure of the algorithm.) Some of our test problems were quite difficult, involving extremely distorted polygons like those shown later in the paper; such problems sometimes required several hundred evaluations of the functions (4.3). The only cases where either HYBRD or Schnabel's code failed eventually to find a solution involved severe crowding in regions that were elongated in more than one direction, as in Fig. 1(c). Though for individual problems there were sometimes wide variations in the number of iterations required by the different routines, all three gave fairly similar performance when averaged over a number of different cases.

Dias [5] and Bjørstad and Grosse [2] have used other nonlinear equations packages for Schwarz-Christoffel problems, with similar results.

In the context of this section, Davis' algorithm (4.1) can be thought of as an approximate Newton iteration in which an approximate Jacobian is estimated from geometrical considerations; our second observation in § 4.1 above amounts to the statement that sometimes this approximation may fail to yield a descent direction.

The secant algorithms converge superlinearly once they are near the solution (see [4]), but for difficult problems they may take a long time to get near it. Convergence times seem to be nearly independent of the starting point, a clear indication that we do not have a good algorithm for picking starting points. To examine the typical convergence rate as a function of N, at least two different types of problems should be considered, as shown in Fig. 6. In the first example the geometry becomes progressively more complicated as N increases, while in the second the geometry is roughly constant, and increasing N merely improves the resolution of the curved part of the boundary. (There are better ways to approximate curved boundaries; see [9] and [27].)

In practice, we find that for problems of the first kind the number of iterations required is roughly $O(N)$, whereas for problems of the second kind it is closer to $O(1)$. Since each evaluation of (4.3) requires $O(N^2)$ operations, and the evaluation of the initial Jacobian matrix by finite differences requires N evaluations, the total work required to solve the parameter problem is at least $O(N^3)$ in both cases.[2] For N less than about 50 the Jacobian evaluation is not the dominant factor in the calculation, however, so problems with simple geometries typically display behavior closer to $O(N^2)$.

5. Evaluating Schwarz–Christoffel integrals. The second numerical problem is the evaluation of (3.5). This cannot be done analytically, and is somewhat difficult numerically because of the singularity in the integrand at each prevertex z_j. A robust integration scheme must be able to deal efficiently not only with the endpoint singularities that occur when one of the limits of integration is a prevertex, but also with the nearly singular situation where a prevertex is very close to the interval of integration. The latter case is important when there is significant crowding, and also when a nearby singularity lies on the opposite side of the strip from the interval of interest. Removing every possible singularity analytically would not be worth the trouble, but neither would refining the mesh over the entire interval just to deal with a few difficult segments. In SCPACK, Trefethen [28] used a compound Gauss-Jacobi quadrature algorithm with considerable success. This method outperforms every alternative we have tried, but since it lacks an internal error check, we have sometimes found it helpful to supplement it with more general adaptive quadrature schemes. The first use of general adaptive quadrature routines for Schwarz-Christoffel integrals appears to be that of Dias [5] as late as 1986.

Singularities at the endpoints themselves are more of a nuisance than a problem since they can be directly accounted for by the quadrature algorithm. The first question is whether we have to integrate them at all. In their program for solving the Schwarzian differential equation for circular arc polygons, Bjørstad and Grosse [2] avoid singularities by integrating to the midpoint of each interval instead of to each prevertex. Corner positions are then found by calculating where sides intersect. However, this approach can run into trouble for difficult regions, particularly when the program is far from a solution to the parameter problem. We have seen examples where the initial guess for the prevertices yielded an image polygon with some side lengths incorrect by factors exceeding 10^{16}. In such examples, adjacent corners may become indistinguishable even in double precision. By contrast, integrating (3.5) directly from one corner to the next permits each side length to be determined individually without cancellation problems.

5.1. Adaptive quadrature. Given that we choose to integrate up to singularities, there are a number of methods to choose from. We can either use a quadrature rule that explicitly takes the singularity into account, such as a Gauss-Jacobi or Clenshaw-Curtis formula, or we can attempt to remove the singularity analytically so that a standard quadrature rule can be used. QUADPACK [24] includes routines that take the explicit approach. The most effective of these for our Schwarz-Christoffel problem is QAWS, an adaptive quadrature subroutine that uses a Gauss-Kronrod formula in the interior and a Clenshaw-Curtis formula near the singularities. All of the adaptive QUADPACK routines, however, seem to be written with the assumption that typical integrals will be very difficult. They use very high-order rules and require a large

[2] The linear algebra required by the secant algorithm can be held to $O(N^2)$ per iteration by using secant updates (see [4]), and is typically negligible compared with the cost of evaluation (4.3).

number of integrand evaluations—50 for QAWS—even when no adaptive refinement is necessary. Since many of the integrals involved in any Schwarz-Christoffel problem are not at all difficult, this expense makes QUADPACK less competitive unless high accuracy is required.

When only moderate accuracy (fewer than eight decimal places) is required in the evaluation of (3.5), we have obtained better performance by using singularity removal along with QUANC8, a simple adaptive routine described in [13]. QUANC8, based on the 8-panel Newton-Cotes formula, is more efficient than the routines in QUADPACK when many of the integrals are well behaved. For solving the parameter problem the 8-panel rule seems to be a good compromise between accuracy and speed, although modified versions of QUANC8 based on lower-order formulas are better for applications involving shorter intervals of integration, e.g., graphics. The key point is that for efficiency an integrator must solve simple problems quickly, whereas for robustness it must include an internal error check and must be able to adaptively refine its mesh if necessary.

5.2. Compound Gauss–Jacobi quadrature. The problem with the adaptive integrators described above is that they do not use all of the available information. Their algorithmic decisions are based solely on the observed behavior of the integrand, whereas in Schwarz-Christoffel problems we know the precise position and strength of every singularity before integration begins. Compound Gauss-Jacobi quadrature is a compromise between fixed-rule algorithms, which are unsatisfactory due to nearby singularities, and fully adaptive algorithms, which are extremely dependable but relatively slow. The idea is to use a Gauss-Jacobi formula on each interval that ends at a singularity and an ordinary Gauss formula on all other intervals, with the rather arbitrary requirement that no outside singularity may lie closer to any interval than half the length of that interval [28]. In our program we implement this by splitting any interval that is too close to a singularity in half recursively, repeating as necessary until every interval of integration is short enough to be acceptable. The nodes and weights for the Gauss-Jacobi quadrature rules are calculated using the routine GAUSSQ by Golub and Welsch [16]; we have found experimentally that the number of accurate decimal places in the solution is approximately the same as the number of nodes used on each interval. The primary drawback of the method is that this is entirely an empirical bound.

In our computations the compound Gauss-Jacobi method has outperformed adaptive rules by a factor of at least 2. There are several ways in which it could be improved—for example, by taking the strengths as well as the positions of outside singularities into account. Perhaps theorems could be developed to establish that a suitably defined compound Gauss-Jacobi algorithm is guaranteed to be successful; in the meantime, a virtually foolproof error bound can be obtained if desired by switching to a high-accuracy adaptive integrator at the end of the solution of the parameter problem. On the other hand, since graphics do not require high accuracy, we have also found it efficient to switch to a low-accuracy adaptive method, based on Simpson's rule, for plotting the final map. Other low-accuracy integration formulas suitable for Schwarz-Christoffel mapping are described in a recent paper by Floryan and Zemach [12].

5.3. Singularity removal. When using a quadrature package that does not explicitly take singularities into account, it is necessary to remove endpoint singularities from the integrand analytically. There are two main methods for removing singularities, and we have found it difficult to pick one as a favorite. If for simplicity we place the singularity at 0, the problem is to integrate a function $f(x)=x^\gamma g(x)$, where g is analytic at 0 and $\gamma>-1$, on an interval $(0, X)$. Expanding $f(x)$ in a power series about 0, we obtain

$$f(x)=x^\gamma g(0)+x^{\gamma+1}g'(0)+\tfrac{1}{2}x^{\gamma+2}g''(0)+O(x^{\gamma+3}). \tag{5.1}$$

The first method for removing the singularity is simply to subtract off the leading terms of (5.1), which can be integrated analytically, and to use a numerical integrator only on the relatively well-behaved remainder. With just the first term removed the remainder may have an infinite slope at 0, which is still enough to cause serious trouble for integrators that assume polynomial behavior. With the first and second terms removed, though, most polynomial integrators perform quite well near the singularity. The use of this two-term subtraction for Schwarz-Christoffel problems dates back at least to Kantorovich and Krylov [20].

The second standard method is to find a change of variables $x=t^\alpha$ such that the integrand is well behaved when expressed in terms of t. In general, we want to choose α so that for $t\approx 0$, $f[x(t)]\,d[x(t)]$ will behave like $t^\beta\,dt$ for some small nonnegative integer β. A little algebra gives $\alpha=(\beta+1)/(\gamma+1)$. The new integrand is then $\alpha t^{\alpha-1}f(t^\alpha)$, which indeed has the expected leading-order behavior near $x=0$. Dias [5] used this method with $\beta=0$, which works quite well for $-1<\gamma\leqq 0$. However, we have found that polynomial integrators still have trouble with the transformed integrand when $\gamma>0$. To see why this happens, let us expand f again and look at what the change of variables does to the higher-order terms:

$$f(x)=x^\gamma(g_0+g_1x+g_2x^2+O(x^3)), \tag{5.2}$$

$$\begin{aligned} f(t^{(\beta+1)/(\gamma+1)}) &= g_0t^{\gamma(\beta+1)/(\gamma+1)}+g_1t^{(\gamma+1)(\beta+1)/(\gamma+1)} \\ &\quad +g_2t^{(\gamma+2)(\beta+1)/(\gamma+1)}+O(t^{(\gamma+3)(\beta+1)/(\gamma+1)}), \end{aligned} \tag{5.3}$$

$$\begin{aligned} t^{(\beta-\gamma)/(\gamma+1)}f(t^{(\beta+1)/(\gamma+1)}) &= g_0t^\beta+g_1t^{\beta+(\beta+1)/(\gamma+1)} \\ &\quad +g_2t^{\beta+2(\beta+1)/(\gamma+1)}+O(t^{\beta+3(\beta+1)/(\gamma+1)}). \end{aligned} \tag{5.4}$$

Note that the g_1 term can have an infinite-slope singularity when $\beta=0$ and $\gamma>0$. The obvious solution to the problem is to use a larger β, but there is a trade-off involved since the resulting large value of α makes the integrand evaluation points cluster near the singularity, so that the adaptive integrator must work harder at the other end of the interval. We have found empirically that using $\beta=0$ for $\gamma<-0.35$, $\beta=1$ otherwise, tends to give the best results, which are slightly better than those obtained using the method of Kantorovich and Krylov.

6. Mapping rectangles to closed polygons.

When the target domain is a closed polygon rather than an infinite channel, it is often appropriate to take an elongated rectangle as the standard region rather than an infinite strip.[3] The aspect ratio of the rectangle will be equal to the conformal modulus ($\approx$electrical resistance) of the original polygon with its four distinguished vertices. We calculate this conformal map by mapping first from the rectangle to the strip by means of an elliptic function, then from the strip to the polygon by the strip transformation.

The function $s(z)=(1/\pi)\log \operatorname{sn}(z\,|\,m)$ maps a rectangle onto a strip of width 1, sending the corners $-K$, K, $K+iK'$, and $-K+iK'$ of the rectangle to the points i, 0, L, and $L+i$, respectively, where K, K', and L are all functions of m (Fig. 7). It seems reasonable to require each corner of the rectangle to map to a vertex of the polygon,

[3] For other approaches to conformal mapping onto rectangles, see [14] and [23].

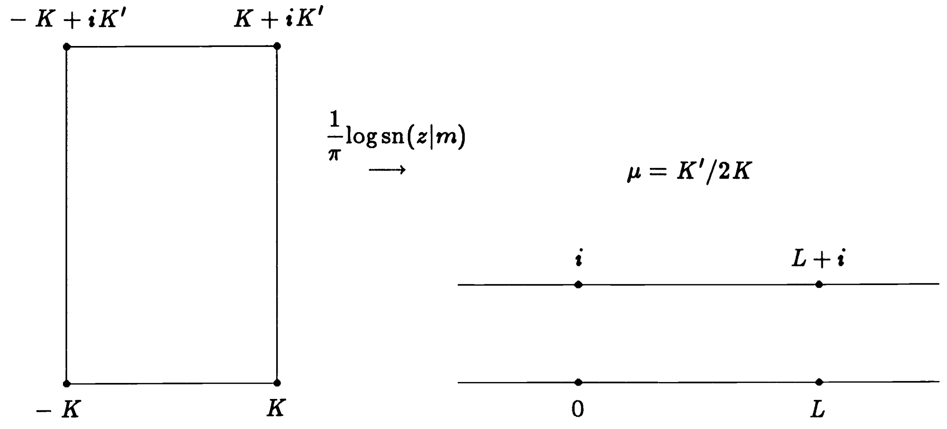


FIG. 7. *Conformal map from a rectangle to a strip.*

and in many applications this is called for since the conformal modulus is required. However, the formulation of the parameter problem that we used in § 4 would not permit this since, in general, no two prevertices have the same real part. We can avoid this difficulty, though, since for a closed polygon we no longer need to specify the images of the ends of the strip. The two extra degrees of freedom thus obtained can be used instead to fix z_N^* at i and require that the two rightmost prevertices have the same real part. Solving the parameter problem with these restrictions thus gives us an appropriate value for L, from which m, K, and K' can be calculated using the known properties of elliptic functions.

In formulating the parameter problem for this version of the strip transformation, we again use the constants A and B in (3.5) to send w_1 and w_N to their correct positions. A suitable set of unconstrained independent variables, corresponding to the new normalization is

$$x_j = \begin{cases} \log(z_{j+1}-z_j), & 1 \leqq j \leqq M-2, \\ \frac{1}{2}[\log(z_M - z_{M-1}) + \log(z_{M+1}-z_{M+2})], & j = M-1, \\ \log(z_{j+2}-z_{j+3}), & M \leqq j \leqq N-3. \end{cases} \tag{6.1}$$

Note that there are only $N-3$ independent variables, corresponding to the fact that now only $N-3$ side length conditions are required to determine the shape of the closed polygon:

$$F_j = \begin{cases} \log \left| \dfrac{w_{j+1}-w_j}{w_{j+1}^*-w_j^*} \right|, & 1 \leqq j \leqq k-2, \\ \log \left| \dfrac{w_{j+2}-w_{j+3}}{w_{j+2}^*-w_{j+3}^*} \right|, & k-1 \leqq j \leqq N-3. \end{cases} \tag{6.2}$$

Here w_k is the "omitted corner"—if $N-1$ corners of the polygon and all of the angles are known, then the position of the remaining corner is determined and cannot be specified separately. The side between w_1 and w_N is fixed by the constants A and B, and the two sides that intersect at w_k do not enter into the parameter problem, so exactly $N-3$ side lengths are sufficient to determine the shape of the polygon.

The best choice of w_k is problem-dependent, and an improper choice can make the system of nonlinear equations much more difficult to solve. In Fig. 8, for example, w_7 and w_8 would both be poor choices for w_k. Since the two sides that intersect at w_7 are collinear, it is not possible to determine the position of the corner by finding the

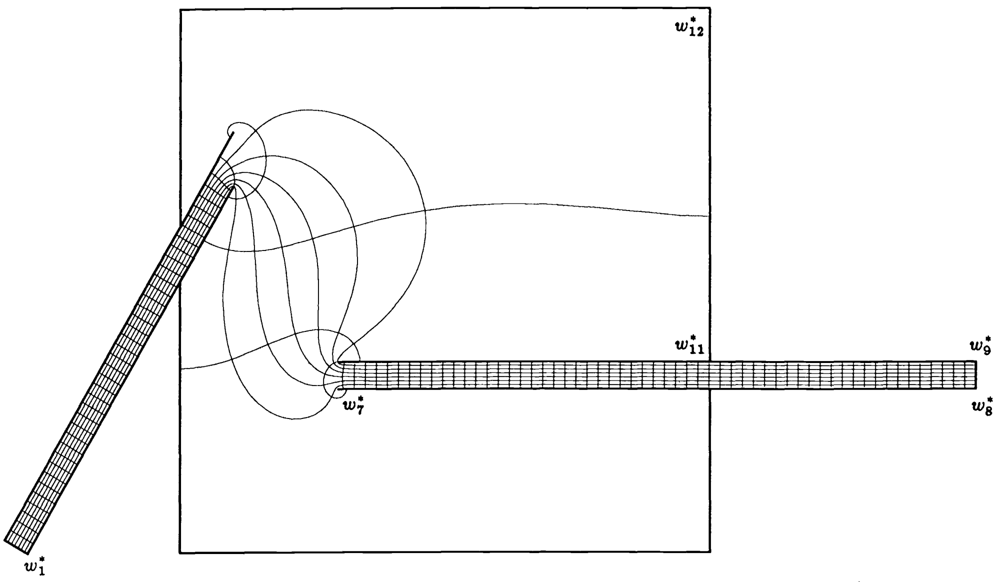


FIG. 8. *Conformal map of a rectangle onto a distorted polygon* ($\mu \approx 41.812465$).

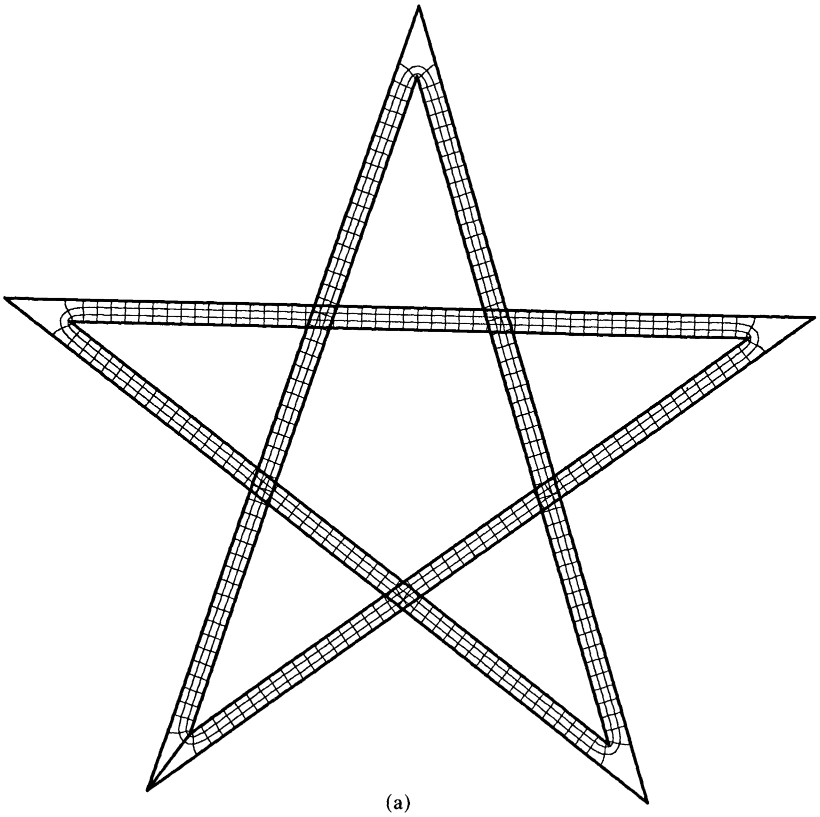
intersection of the sides. To see why w_8 is bad, picture a slight distortion of this polygon in which the distance between w_{11} and w_{12} is increased and the horizontal tube is correspondingly narrowed. This narrowing increases the aspect ratio of the tube dramatically, so z_8 and z_9 would be far to the right of their correct positions. This will only affect the dependent variables to a small degree, though, if the distance between w_8 and w_9 is not included in (6.2). The system of nonlinear equations will therefore be poorly scaled, and the algorithm will probably take much longer to converge. A much better choice in this case would be w_{12}, which does not introduce any scaling problems. Several other choices would be equally good, and in fact it is sometimes helpful to change w_k in the middle of the solution process if the nonlinear equations algorithm is making slower progress.[4]

At present, our code leaves the choice of w_k up to the user, but we may be able to automate this in the future. Figure 9(a)-(b) shows two examples of regions for which no choice of w_k is very good: the only way to make these problems well scaled would be to devise a radically different set of dependent variables.[5] Note that the self-intersecting nature of the first domain does not cause any difficulties; the problem results from the fact that changing almost any side length slightly can greatly alter the conformal modulus of the polygon.

7. Variations. The two problem formulations described in §§ 4 and 6 illustrate some of the choices that can be made with the strip transformation, but by no means do they exhaust the possibilities. With the channel mapping, for instance, it is not

[4] The system of equations for the channel map can be altered in a similar manner, but the presence of the angle in (4.3) raises complications. One must be careful not to create a set of nonlinear equations with more than one solution.

[5] It is still possible to calculate accurate conformal moduli for these polygons, however, even though the parameter problems are slow to converge. So that others may reproduce these examples if they wish, in the star the width of the strip is 1/20 of the radius of the circumscribing circle, and in the spiral the width of the strip is exactly half the width of the complementary white space. The regions shown in Figs. 8 and 10(b) have corners exactly: $\{(-.2887, 0.), (.1, .6732), (0., .4999), (0., 0.), (1., 0.), (1., .2999), (.3, .3), (1.5, .3), (1.5, .35), (.3, .35), (1., .3501), (1., 1.), (0., 1.), (0., .6001), (.1, .7732), (-.3320, .025)\}$ and $\{(0., 8.), (1.8, 8.), (1.8, 0.), (13., 0.), (14., 0.), (14., 1.), (2., 1.), (2., 10.), (0., 10.)\}$, respectively.


(a)

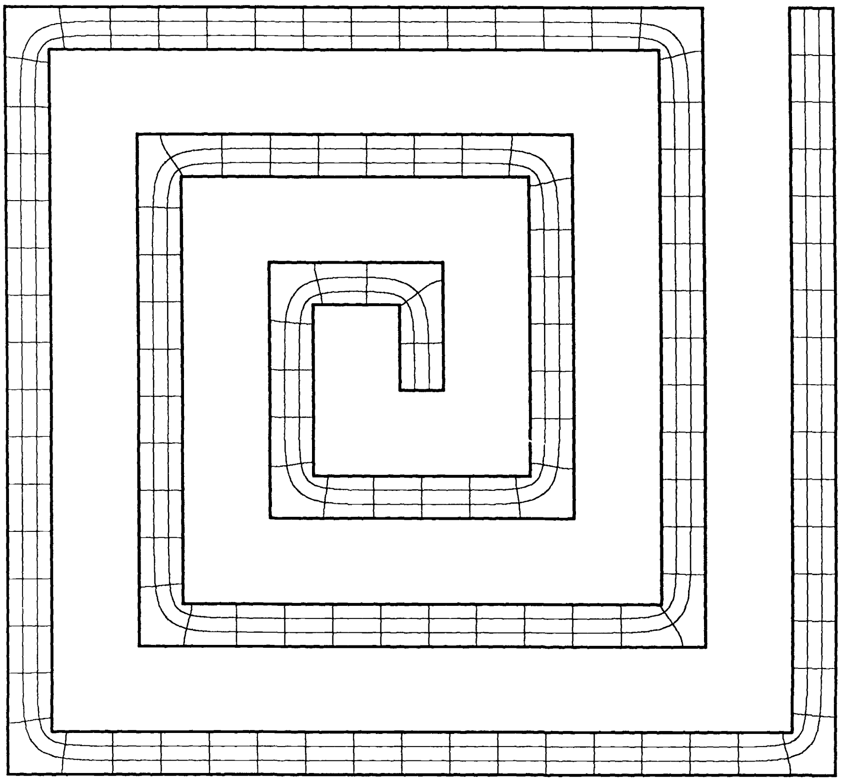

(b)

FIG. 9. (a) *Conformal map of a rectangle onto a star* ($\mu \approx 163.28151$). *Note that both ends of the polygon meet at the lower-left corner.* (*b*) *Conformal map of a rectangle onto a finite spiral* ($\mu \approx 132.70454$).

necessary for either of the divergence angles θ_- and θ_+ to be positive; the channel may be bounded. Figure 10(a) shows an example where both are negative; physically this could represent an electromagnetic problem with point charges or currents, or a fluids problem with a source and a sink. (This region has significant secondary crowding effects near the point of the barb, due to the extremely acute angle there. It is impossible to plot streamlines much closer to the point than those shown using the integration methods we have described. The other angles are wide enough to avoid this difficulty, though there is always some degradation of accuracy near an outward-pointing corner.)

With the rectangle mapping there is no reason why each end of the rectangle must map onto a single side of the polygon, and there are many possible ways to modify

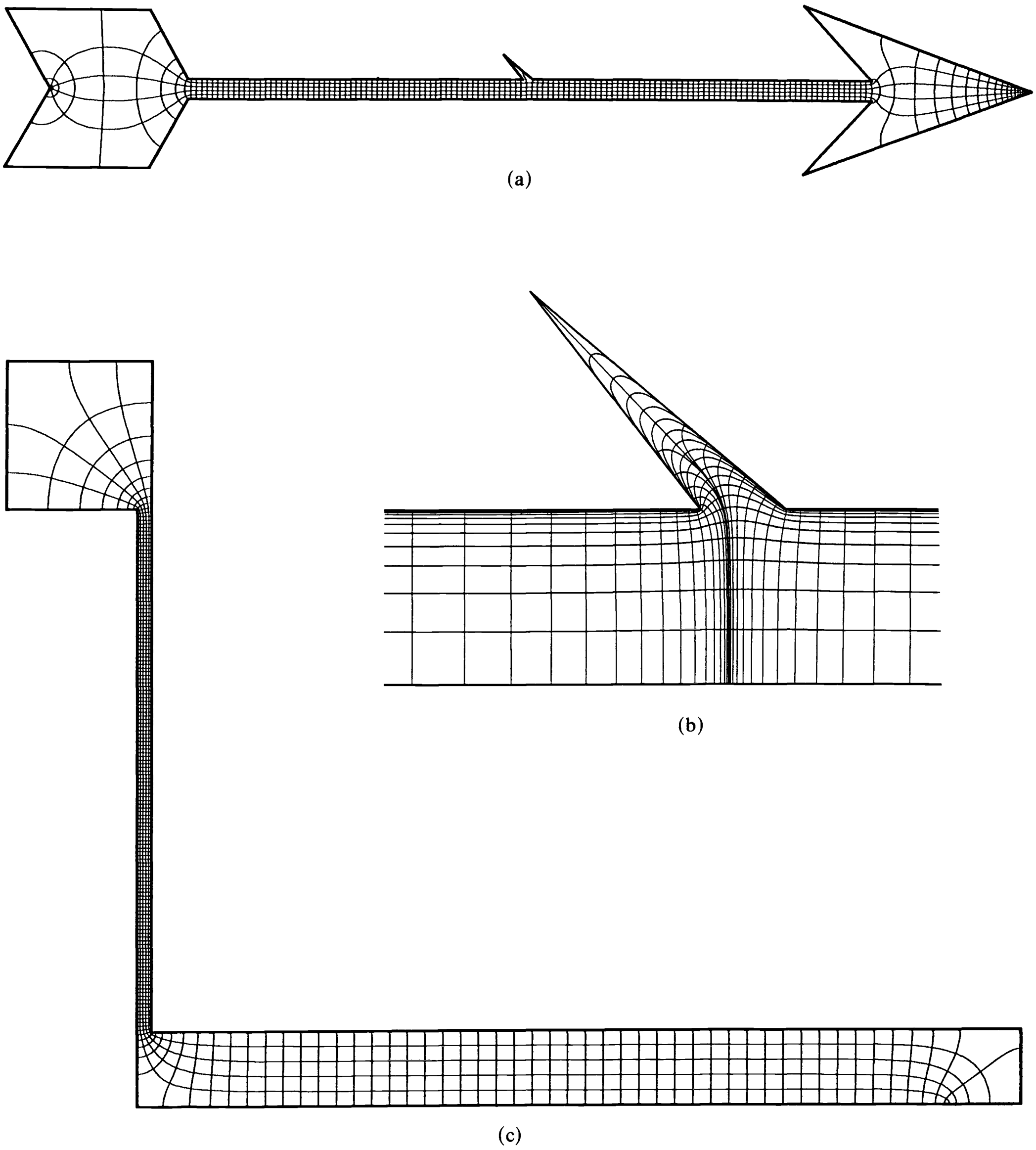


FIG. 10. (a) *A channel map with converging ends*; (b) *enlargement of the barb in Fig.* 10(a); (c) *a rectangle map onto a polygonal conductor* ($\mu \approx 49.436547$).

the given formulation to deal with unusual cases. In Fig. 10(c), for example, we have fixed w^*_{N-1} at i instead of w^*_N, and introduced an additional vertex w_4 with a turning angle of 0. Problems like this one could arise in integrated circuit design.

Other straightforward variations of the method described in this paper include vertices at infinity, the exterior map for a polygon, the map from a semi-infinite strip to a channel bounded at one end, and various generalized parameter problems as those described in [30] and [8].

Figure 11 shows a more extreme variation, an infinite logarithmic spiral. To permit our program to run to completion in a finite time, we truncated the infinite product in (3.5) by ignoring corners more than three turns away from the point of interest. This approach yields accurate results since the effect of each singularity decays exponentially along the strip. The parameter problem for this example is also quite simple, since the domain is self-similar; we omit the details. A similar formulation was used by Floryan [10] to map periodic channel configurations. The ideas involved in this example might possibly be extended to permit the mapping of more complicated fractal domains, which would have applications, for example, in the study of diffusion-limited aggregation.

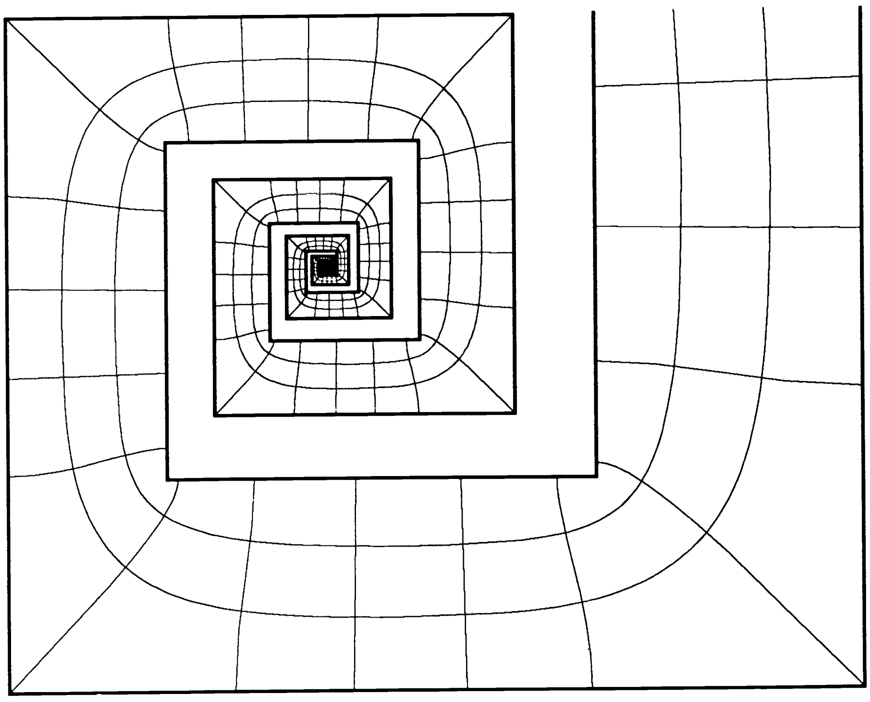


FIG. 11. *Conformal map onto an infinite spiral.*

Sridhar and Davis [3], [27], Floryan [9], and Hoekstra [19] have all described another variation of channel maps for approximating curved boundaries, based on formulas dating back at least to Woods [33]. Our implementation does not currently include this variation, but it would certainly be a valuable addition to any future software package for calculating Schwarz-Christoffel maps.

8. Conclusion. To summarize the central point of this paper: Conformal maps of highly elongated polygons should be based on a Schwarz-Christoffel formula for an infinite strip, not a disk or a half plane. Many of the polygons that arise in applications are of this type—perhaps most.

The mathematics of the Schwarz-Christoffel formula for an infinite strip is not new. What is new here is, first, the proposal that such a formula sould be used even when the polygon is bounded rather than an infinite channel, and second, an algorithm for numerical integration and solution of the parameter problem that can reliably and efficiently compute conformal maps to high accuracy (e.g., 8 or 12 digits, except in regions subject to secondary crowding) even for extremely elongated polygons (e.g., with aspect ratios in the hundreds or thousands). The elements of this algorithm are adapted from the SCPACK package for mapping the unit disk.

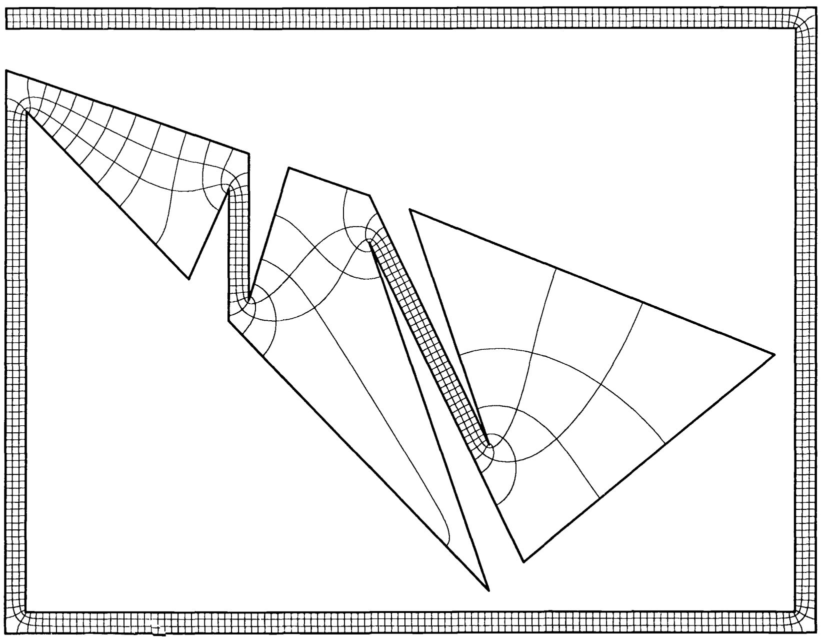


FIG. 12. *A polygon with* 23 *sides* ($\mu \approx 156.6241139$).

We conclude with a final example. The rather difficult 23-sided polygon of Fig. 12 was mapped from a rectangle, to roughly 10-digit precision, in about one hour on a Sun 3/50 with an MC68881 floating-point coprocessor. Most of this time was spent in solving the parameter problem, and since the final convergence is quite rapid, the time is nearly independent of the required accuracy. The same calculation on a supercomputer would require only a few seconds.

Appendix. Evaluation of elliptic functions. The use of the rectangle mapping described in § 6 requires the efficient evaluation of the function $s(z) = (1/\pi) \log \operatorname{sn}(z \mid m)$ over a wide range of values of z and m. All of the formulas we use for these computations are well known and available in standard references, but since there are several different asymptotic regimes involved, it seems appropriate to give a brief summary of our methods here. For more information about any of the following material, see [1].

The parameter m of the elliptic function $\operatorname{sn}(z \mid m)$ decreases exponentially with $K'/2K$, the conformal modulus of the rectangle. For highly elongated regions, therefore,

m may be less than the underflow limit of many computers. First we will deal with the case where m is reasonably large; in our own program this means $m > 10^{-30}$.

To start with, the value of L is determined by the solution to the parameter problem for the strip transformation, so $m = e^{-2\pi L}$ is known immediately. K and K' are complete elliptic integrals with parameter m, and can be readily found by means of the arithmetic-geometric mean (AGM) method. To calculate K we first take

(A.1) $$a_0 = 1, \qquad b_0 = \sqrt{m_1}, \qquad c_0 = \sqrt{m};$$

$m_1 = 1 - m$ is called the complementary parameter. The AGM iteration, defined by the formulas

(A.2) $$a_i := \tfrac{1}{2}(a_{i-1} + b_{i-1}), \qquad b_i := (a_{i-1}b_{i-1})^{1/2}, \qquad c_i := \tfrac{1}{2}(a_{i-1} - b_{i-1}),$$

is then carried out until at the Nth step c_N is negligible to the required accuracy. K is then equal to $\pi/2a_N$; to find K' the same procedure is followed with m and m_1 interchanged.

We need to evaluate $\mathrm{sn}\,(z\,|\,m)$ at points inside the rectangle with corners $-K$, K, $K + iK'$, $-K + iK'$. This function has a pole at iK', but we avoid any difficulties there by using the identity

(A.3) $$\mathrm{sn}\,(z\,|\,m) = \frac{-1}{m^{1/2}\,\mathrm{sn}\,(iK' - z\,|\,m)}$$

whenever $\mathrm{Im}\,(z) > K'/2$. For $\mathrm{Im}\,(z) \leqq K'/2$ we can approximate $\mathrm{sn}\,(z\,|\,m)$ by

(A.4) $$\mathrm{sn}\,(z\,|\,m) \sim \sin(z) - \frac{1}{4}m[z - \sin(z)\cos(z)]\cos(z)$$

when m is small enough; the relative accuracy of this formula is $O(m)$ for $\mathrm{Im}\,(z) \leqq K'/2$. If m is not sufficiently small we can reduce it by applying the descending Landen transformation as many times as necessary:

(A.5) $$\mu = \left(\frac{1 - m_1^{1/2}}{1 + m_1^{1/2}}\right)^2,$$

(A.6) $$v = \frac{z}{1 + \mu^{1/2}},$$

(A.7) $$\mathrm{sn}\,(z\,|\,m) = \frac{(1 + \mu^{1/2})\,\mathrm{sn}\,(v\,|\,\mu)}{1 + \mu^{1/2}\,\mathrm{sn}^2\,(v\,|\,\mu)}.$$

The effect is to replace m and z by μ (not to be confused with the conformal modulus) and v, where $\mu \sim m^2/16$ and $v \sim z$. To avoid cancellation errors, (A.5) should be evaluated via a power series when m is less than about 10^{-3}.

When $L > 11$, i.e., $m < 10^{-30}$, we use a different set of asymptotic formulas to avoid possible underflow of m or overflow of $\mathrm{sn}\,(z\,|\,m)$. K and K' are calculated via the approximations

(A.8) $$K \sim \frac{\pi}{2} \quad \text{and} \quad K' \sim \pi L + \log 4,$$

which are accurate to $O(m)$. For $\mathrm{Im}\,(z) \leqq K'/2$ the approximation $\mathrm{sn}\,(z\,|\,m) \sim \sin(z)$ has a relative accuracy of $O(m^{1/2})$, and $\log(\sin z)$ can be expanded to give

(A.9) $$s(z) \sim \frac{1}{\pi}\left\{-iz + \log\left[\frac{1}{2i}(e^{2iz} - 1)\right]\right\}.$$

When $\operatorname{Im}(z) > K'/2$ the identity (A.3) leads to the similar formula

$$s(z) \sim L + i + \frac{1}{\pi}\left\{iu - \log\left[\frac{1}{2i}(e^{2iu}-1)\right]\right\}, \tag{A.10}$$

where $u = iK' - z$.

By the methods described here we can evaluate $s(z) = (1/\pi)\log \operatorname{sn}(z\,|\,m)$ to close to full precision (around 15 decimal places in our calculations), for all z in the fundamental rectangle, for a range of parameters roughly $e^{-2\pi/\varepsilon} \ll m < \frac{1}{2}$, that is, $O(1) \leqq L \ll \varepsilon^{-1}$, where ε is the machine precision.

REFERENCES

[1] M. Abramowitz and I. Stegun, eds., *Handbook of Mathematical Functions*, National Bureau of Standards, Washington, D.C., 1964.

[2] P. Bjørstad and E. Grosse, *Conformal mapping of circular arc polygons*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 19-32.

[3] R. T. Davis, *Numerical methods for coordinate generation based on Schwarz-Christoffel transformations*, Proc. 4th Amer. Inst. Aero. Astro. Computational Fluid Dynamics Conference, Williamsburg, VA, 1979.

[4] J. E. Dennis Jr. and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[5] F. Dias, *On the use of the Schwarz-Christoffel transformation for the numerical solution of potential flow problems*, Ph.D. thesis, University of Wisconsin, Madison, WI, 1986.

[6] J. J. Dongarra and E. Grosse, *Distribution of mathematical software via electronic mail*, Commun. ACM, 30 (1987), pp. 403-407.

[7] M. Dubiner, *Theoretical and numerical analysis of conformal mapping*, Ph.D. thesis, Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA, 1981.

[8] A. R. Elcrat and L. N. Trefethen, *Classical free-streamline flow over a polygonal obstacle*, J. Comput. Appl. Math., 14 (1986), pp. 251-265.

[9] J. M. Floryan, *Conformal-mapping-based coordinate generation method for channel flows*, J. Comput. Phys., 58 (1985), pp. 229-245.

[10] ———, *Conformal-mapping-based coordinate generation method for flows in periodic configurations*, J. Comput. Phys., 62 (1986), pp. 221-247.

[11] J. M. Floryan and C. Zemach, *Schwarz-Christoffel mappings: A general approach*, J. Comput. Phys., 72 (1987), pp. 347-371.

[12] ———, *Quadrature rules for singular integrals with application to Schwarz-Christoffel mappings*, J. Comput. Phys., 75 (1988), pp. 15-30.

[13] G. E. Forsythe, M. A. Malcolm, and C. B. Moler, *Computer Methods for Mathematical Computations*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

[14] D. Gaier, *Ermittlung des konformen Moduls von Vierecken mit Differenzenmethoden*, Numer. Math., 19 (1972), pp. 179-194.

[15] A. F. Ghoniem and Y. Gagnon, *Vortex simulation of laminar recirculating flow*, J. Comput. Phys., 68 (1987), pp. 346-377.

[16] G. H. Golub and J. H. Welsch, *Calculation of Gaussian quadrature rules*, Math. Comp., 23 (1969), pp. 221-230.

[17] L. Greengard, *Potential flow in channels*, SIAM J. Sci. Statist. Comput., submitted.

[18] P. Henrici, *Applied and Computational Complex Analysis*, Vol. 3, John Wiley, New York, 1986.

[19] M. Hoekstra, *Coordinate generation in symmetrical interior, exterior or annular* 2-D *domains, using a generalized Schwarz-Christoffel transformation*, in Numerical Grid Generation in Computational Fluid Mechanics, J. Hauser and C. Taylor, eds., Pineridge Press, Swansea, United Kingdom, 1986.

[20] L. V. Kantorovich and V. I. Krylov, *Approximate Methods of Higher Analysis*, P. Noordhoff, Groningen, the Netherlands, 1958.

[21] R. Menikoff and C. Zemach, *Methods for numerical conformal mapping*, J. Comput. Phys., 36 (1980), pp. 366-410.

[22] J. J. Moré, B. S. Garbow, and K. E. Hillstrom, *User guide for* MINPACK-1, Argonne National Laboratory Report, ANL-80-74, Argonne, IL, 1980.

[23] N. PAPAMICHAEL, C. A. KOKKINOS, AND M. K. WARBY, *Numerical techniques for conformal mapping onto a rectangle*, J. Comput. Appl. Math., 20 (1987), pp. 349-358.

[24] R. PIESSENS, E. DE DONCKER-KAPENGA, C. W. ÜBERHUBER, AND D. K. KAHANER, QUADPACK: *A Subroutine Package for Automatic Integration*, Springer-Verlag, Berlin, New York, 1983.

[25] M. J. D. POWELL, *A Fortran subroutine for solving systems of non-linear algebraic equations*, Tech. Report AERE-R. 5947, Harwell, England, 1968.

[26] K. REPPE, *Berechnung von Magnetfeldern mit Hilfe der konformen Abbildung durch numerische Integration der Abbildungsfunktion von Schwarz-Christoffel*, Siemens Forsch. u. Entwickl. Ber., 8 (1979), pp. 190-195.

[27] K. P. SRIDHAR AND R. T. DAVIS, *A Schwarz-Christoffel method for generating two-dimensional flow grids*, J. Fluids Eng., 107 (1985), pp. 330-337.

[28] L. N. TREFETHEN, *Numerical computation of the Schwarz-Christoffel transformation*, SIAM J. Sci. Statist. Comput., 1 (1980), pp. 82-102.

[29] ———, SCPACK User's Guide, Numerical Analysis Report 89-2, Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA, 1989.

[30] ———, *Analysis and design of polygonal resistors by conformal mapping*, J. Appl. Math. Phys., 35 (1984), pp. 692-704.

[31] ———, ED., *Numerical Conformal Mapping*, North-Holland, Amsterdam, 1986.

[32] ———, *Schwarz-Christoffel mapping in the* 1980'*s*, Numerical Analysis Report 89-1, Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA, 1989.

[33] L. C. WOODS, *The Theory of Subsonic Plane Flow*, Cambridge University Press, United Kingdom, 1961.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 5, pp. 950-974, September 1990

NUMERICAL STUDY OF THE MECHANISMS FOR INITIATION OF REACTING SHOCK WAVES*

A. J. MAJDA† AND V. ROYTBURD‡

Abstract. Experimental evidence strongly supports the role of exothermic reaction centers as robust structures of fundamental importance in both the secondary stages of transition to detonation and the initiation of detonation. Here it is demonstrated through careful high resolution numerical computation that the complete range of experimental phenomena observed in the transient behavior following the formation of a single exothermic reaction center can be found in solutions of the inviscid reactive Euler equations in a single space dimension with simplified one-step irreversible Arrhenius kinetics. Furthermore, two new mechanisms for rapid initiation of detonation through the resonant interaction of exothermic reaction centers are demonstrated.

Key words. reacting shock waves, exothermic reaction centers, transition to detonation

AMS(MOS) subject classifications. 76L05, 80A32, 65P05

1. Introduction.

Experiments in gaseous phases reveal extremely complex interactions among turbulence, chemistry, and strong compressibility in such important practical problems as the transition to detonation [1], [2] or direct initiation of detonation [3]. Different but also rather complex effects are observed in the transition to detonation in condensed phases as well [4]. Despite the tremendous complexity of these phenomena, Oppenheim and coworkers (see [5], [6], for example) have emphasized the role of exothermic reaction centers as a robust structure of fundamental importance in the secondary stages of transition to detonation; such exothermic reaction centers also form approximate initial data for the experiments regarding initiation of detonation. A one-dimensional idealization of an exothermic reaction center is a nonreactive inert square-pulse in the temperature profile at constant volume as depicted in the graph in Fig. 1 below.

Thus, in a one-dimensional reactive mixture with simplified one-step chemistry the initial data for an idealized exothermic reaction center are given by an overall constant density with the mass fraction of reactant having the value one, whereas the temperature has the structure depicted in Fig. 1.

One principal goal of this paper is to demonstrate through careful high resolution numerical computations that the complete range of experimental phenomena observed in the transient behavior following the formation of a single exothermic reaction center can be found qualitatively in solutions of the inviscid reactive Euler equations in a single space dimension with simplified one-step irreversible Arrhenius kinetics. The second objective of this paper is to demonstrate potential new mechanisms for rapid initiation of detonation through the resonant interaction of exothermic reaction centers. Here is a brief summary of the contents of this paper. In § 2, we present the reactive Euler equations, the appropriate nondimensional scalings used throughout the paper, and a description of the numerical schemes. The routes to initiation from a single exothermic reaction center are presented in § 3. There we describe high resolution

* Received by the editors May 10, 1989; accepted for publication July 25, 1989.

† Department of Mathematics and Program in Applied and Computational Mathematics, Princeton University, Princeton, New Jersey 08544. The work of this author was partially supported by Army Research Office grant ARO DAAL03-89-K-0013 and by Office of Naval Research grant ONR N00014-89-J-1044.

‡ Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, New York 12180. The work of this author was partially supported by National Science Foundation grant DMS-8603506.

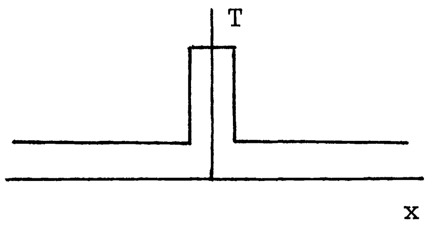


FIG. 1. *The temperature profile in an idealized one-dimensional exothermic reaction center.*

numerical simulations of the time-dependent behavior of solutions of the reactive Euler equations with initial data consisting of a single exothermic reaction center where we vary the height and width of the initial temperature profile. We exhibit transient behavior qualitatively illustrating the subcritical, supercritical, and critical experimental regimes of initiation. (See the review article by Lee [3] for this terminology.) Thus, our numerical calculations exhibit the phenomena of strong initiation, strong failure, initiation through weak ignition effects, and also failure even though secondary weak ignition effects occur. We also display the effects of expanding geometry for initiation by comparing calculations in one dimension with those in a cylindrical geometry. We use the same initial data yielding initiation through weak ignition in a single space dimension for a computation in cylindrical geometry and observe strong failure due to the attenuation effects of the expanding geometry. In § 4, we describe the results of numerical experiments illustrating the role of the resonant effects of multiple exothermic reaction centers in accelerating the transition to detonation. In the first example we consider initial data with multiple hot spots that individually lead to failure; we demonstrate that resonant wave interaction between the hot spots leads to initiation of the detonation. In the second example, we show that high frequency small amplitude perturbations over a constant background state lead to dramatic shortening of the induction time when compared with the homogeneous induction time. Furthermore, these resonant effects are increasingly prominent when parameters in the equations of state are varied to more closely approximate condensed phases. These direct simulations provide supporting evidence for the theoretical explanation of dramatic enhanced combustion in condensed phases recently developed by Almgren, Majda, and Rosales [7]. In that work, quantitative simplified asymptotic equations are derived and solutions are found which indicate that the tremendously enhanced combustion in condensed phases is driven by resonant nonlinear acoustics on rapid timescales. The results of numerical calculations that we present at the end of § 4 are only the first direct simulations on this problem and further detailed numerical studies should be done in the future.

During the last 20 years, there has been considerable activity in the numerical simulation of the initiation and propagation of detonations (see, for example, [8]-[13]). Besides the above-mentioned phenomena we document in this paper, two main features distinguish our numerical approach from most of the previous work:

(1) We focus on details of the transient behavior of the spatial distribution of the state variables;

(2) We pay particular attention to the choice of numerical methods.

In most previous work rather dissipative and/or oscillatory numerical methods were used. Strong purely numerical artifacts in such methods that falsify the physics have been documented and systematically analyzed recently [14]-[15]. A notable exception to the above statements is the work of Oran et al. on weak and strong ignition (see [11]-[13]). Through high resolution numerical schemes, this work carefully documents the role of the transient spatial structure with exothermic reaction centers in weak and strong initiation in hydrogen-oxygen systems. The simulations of Oran, Boris, et al. involve the full complex chemistry of hydrogen-oxygen with eight reacting species. As described by Borisov [16], the origin of exothermic reacting centers can involve either the fluid mechanical effects of pressure and entropy waves or the effects of complex chemistry involving fluctuations of activated molecules or radical concentrations and catalytic generation of radicals. Clearly, the origin of the reaction centers in [11] combines both the effects of fluid mechanics and complex chemistry. Our calculations presented in § 3 illustrating weak initiation or weak failure through the effects of weak ignition conclusively demonstrate that purely fluid mechanical effects alone can play the dominant role in weak ignition regimes, since in this paper we have simplified single step irreversible Arrhenius kinetics.

Our choice of a split numerical scheme consisting of the random choice scheme for the hydrodynamics and a separate fractional step for the chemistry is motivated by the careful numerical test cases developed in [14], [15], and [17]. In particular, the random choice method keeps contact discontinuities perfectly sharp and this is essential for the weak ignition effects that we study here. Such a scheme has also been used by Bukiet [18] in other combustion calculations in both one-dimensional and spherical geometry. In 1977 Chorin [19] proposed the attractive idea of using the Chapman-Jouguet theory directly in a Riemann solver with the random choice scheme without a separate fractional step for the chemistry to remove the stiffness of the system. If this scheme is applied to the test problems in § 3, the method always yields initiation of detonation in contrast to the rather subtle effects with failure, weak failure, and weak initiation that we document in § 3. Thus, for the problems we study here, we need to assess the explicit effects of the chemistry in a separate fractional step. Similar remarks apply for the use of front tracking schemes based on the Chapman-Jouguet theory to track reaction waves for the specific initiation problems considered here. On the other hand, it is extremely attractive to use nonreactive shock tracking (see [20], [21]) to study the interaction of shocks and exothermic reactions centers. Bourlioux, Majda, and Roytburd [22] will report on the use of such schemes in the numerical modeling of unstable detonations in the near future.

2. The basic equations of reacting gas flow and the numerical methods.

We make two standard simplifying assumptions. We neglect all the dissipation mechanisms since they are of secondary importance for the initiation of detonation. For the chemical interaction we consider the simplest model: there are only two species present, the reactant and the product, and the reactant is converted to the product by a one-step irreversible chemical reaction governed by Arrhenius kinetics.

Thus, as a basic description of the reacting mixture we take the one-dimensional Euler equations with one-step irreversible chemical reaction:

$$\rho_t+(u\rho)_x=0, \tag{2.1a}$$

$$(\rho u)_t+(\rho u^2+(1/\gamma)p)_x=0, \tag{2.1b}$$

$$(\rho E)_t+(\rho uE+(1/\gamma)up)_x=0, \tag{2.1c}$$

$$(\rho Z)_t+(\rho uZ)_x=-w, \tag{2.1d}$$

where

$$E=\frac{1}{\gamma-1}T+\frac{u^2}{2}+q_0Z, \tag{2.2}$$
$$p=\rho T, \qquad w=A\rho Z\exp\left(-\frac{E^+}{T}\right).$$

In these equations, p, ρ, T, u, E, and Z are, respectively, the gas pressure, density, temperature, velocity, specific energy, and reactant mass fraction. The variables have been rendered dimensionless by referring them to a constant state. Velocity is referred to the frozen acoustic speed at the reference state, and this nondimensionalization leads to the appearance of factors $1/\gamma$ in (2.1). The dimensionless parameters appearing above are the polytropic exponent γ, the heat release parameter q_0, and the activation energy E^+. To avoid the cold boundary difficulty we employ the ignition temperature that is 1°K above the ambient temperature. It should be pointed out that for the activation energies we consider the thermal runaway develops on the timescale much longer than the computational time.

There is an important timescale, the half-reaction time, that is characteristic for the transition to detonation process. The half-reaction time is defined as follows.

Consider the slowest traveling wave solution of (2.1) compatible with the initial data. This wave is the Chapman-Jouguet (CJ) detonation (see [4]). For this traveling wave solution, the reaction rate equation from (2.1) can be rewritten in the form

$$\frac{dZ}{dt}=-AZ\exp\left(-\frac{E^+}{T}\right),$$

along the characteristics $dx/dt=u$. Obviously, the multiplication of the rate factor A by a constant is equivalent to rescaling the time and space variables by the same constant. In all our calculations except those in § 4.2 we scale time and space (with the corresponding change in A) so that *the time unit is the half-reaction time.* This unit is the time required, in the steady solution, for half-completion of the reaction from the instant of the passage of the fluid element through the shock.

Thus, the calculation of the rescaled rate multiplier involves the following:

(1) Finding the traveling CJ solution. The explicit expressions for the state variables as functions of Z can be found through rather involved algebraic manipulations (see Fickett and Davis [4, Chap. 2]);

(2) Finding the half-reaction time:

$$t_{1/2}=\int_{1/2}^{1}\frac{dZ}{AZ}\exp\left(-\frac{E^+}{T(Z)}\right).$$

This yields

$$A_{\text{new}}=At_{1/2}.$$

Such a scaling is quite standard (see, e.g., [9]). By integrating the characteristic equation

$$\frac{dx}{dt}=u,$$

we can find the corresponding spatial interval, the half-reaction length,

$$\Delta_{1/2}=\int_0^{t_{1/2}}u(Z(-Dt))\,dt\approx t_{1/2}(D-\tilde{u}),$$

where D is the CJ detonation speed and $\tilde{u}$ is the post shock particle velocity. It should be noted that $\Delta_{1/2}$ *may vary widely.* For example, for hydrocarbons $\Delta_{1/2}$ ranges from 0.27–0.29 mm (acetylene) to ~25 mm (methane) (see [24]).

For numerical integration of the reactive Euler equations (2.1), a very natural fractional step scheme is used. The algorithm involves two ingredients per timestep. In the first fractional step the hydrodynamic part of the problem is solved. Equations (2.1a)–(2.1c) are advanced by using the uniform sampling method with Z advected as a passive scalar. In the second fractional step the species equation (2.1d) is advanced by explicitly solving the ordinary differential equation (ODE) for the mass fraction given the temperature field from the previous fractional step. Then new temperature and pressure are found from (2.2) through ρ, u, Z, and E and the process is repeated. Computations are performed in the finite interval with reflecting boundary conditions at the origin and ambient Dirichelt boundary conditions at the other end of the interval. The same numerical method was employed in our previous work [17] and a similar method was employed by Bukiet [18]. In § 3, we present some simulations in a cylindrical geometry. We remind the reader that for the spherically/cylindrically symmetric case, the continuity equation (2.1a) acquires a source type term:

$$\rho_t + (\rho u)_x + \frac{k}{x}\rho u = 0,$$

where $k = 0, 1, 2$ for the planar, cylindrical, and spherical cases correspondingly. Numerically, we treat the source term through an additional fractional step that solves the ODE for ρ.

3. Initiation from a single exothermic reaction center. In this section we describe results of several simulations of detonation initiation or failure from a single initial temperature pulse. For all the calculations in this section we keep the nondimensional heat release fixed $q_0 = 10$. We remind the reader that for given upstream conditions, q_0 completely determines values of the state variables at the peak of the traveling Chapman–Jouguet detonation (also called the von Neumann spike; see [4]). For the upstream condition given by quiescent gas and for $q_0 = 10$, the von Neumann pressure peak is 17.2 in the nondimensional units. Other thermodynamic and chemical parameters are also kept fixed for all the calculations $\gamma = 1.4$, $E^+ = 10$ except for one simulation (§ 3.2) where we demonstrate the quenching effects of raising the activation energy. When we report initial temperature magnitudes in the reaction centers, we give them in units relative to the quiescent temperature.

We investigate several scenarios of transition to detonation and for each case we display the numerical results in a uniform format:

(i) The basic trends of the wave development are illustrated on a diagram that represents the pressure at the front of the wave propagating in the quiescent medium as a function of time. For brevity we call this diagram *the pressure history.*

(ii) We carefully select *a few time levels* (usually 3–4) *at which the spatial distributions of the state variables reveal characteristic features of the process under investigation.* These time instances are marked with "o" on the pressure history and for each one of these we display the spatial distributions of pressure p, temperature T, and the reactant mass fraction Z.

A comment on the numerical resolution of our simulations is in order. We performed a variety of rather expensive and complex calculations and it was not feasible to conduct a comprehensive convergence study. Instead, we started each simulation with coarse grid calculations and did mesh refinements (by doubling) until the results of two consecutive calculations were virtually identical. As we might expect, more violent or unstable evolution scenarios require finer meshes to get full resolution with a corresponding increase in expense. Finally, we note that for our choice of parameters *the spatial unit is almost the same as the half-reaction width* $\Delta_{1/2}=0.89$, so the number of zones per unit length essentially corresponds to the number of zones in a half reacting width.

3.1. Strong initiation. The process in this simulation is initiated by a reaction center of temperature eight and width of five spatial units. The transition to detonation here is powerful and rather straightforward.

The pressure history (Fig. 3.1.1) shows an (almost) monotome increase in pressure at the front. The initial increase gradually slows down until $t \sim 15$ when it again

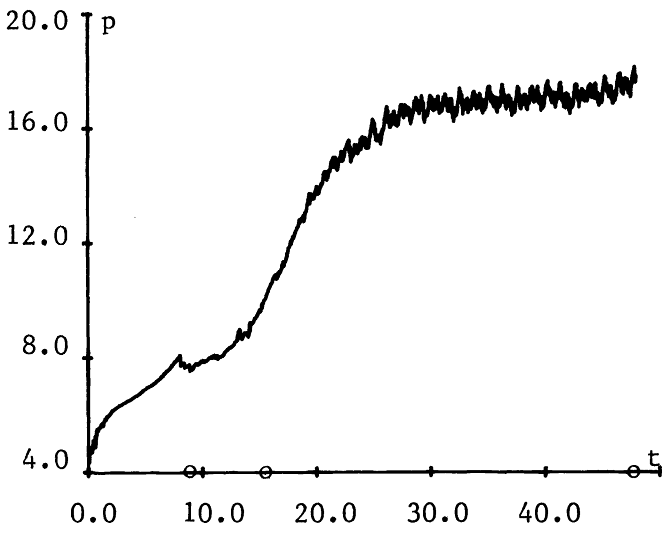


FIG. 3.1.1. *The pressure history for the strong initiation.*

accelerates to reach the CJ value at $t \sim 28$. After that the pressure at the front slightly overshoots the CJ value. We note that noisy oscillations in the peak pressure are characteristic for the uniform sampling method. Comparison of pressure and reactant spatial profiles demonstrates that for *all times* the reaction wave starts immediately at the shock. Thus the shock and reaction waves propagate as a single unit. This precisely corresponds to the supercritical initiation in Lee's terminology [3]. We note that the methods of the present paper were used recently by Kapila and Roytburd [25] to study strong initiation from a smooth initial hot spot.

The slowdown in the pressure increase happens during the development of a secondary reaction center. This effect is not especially significant for the strong initiation that we describe here, but is a crucial "weak ignition" effect that is characteristic of the calculations we present later, so we describe this event in detail. It is apparent from the Z profile at $t=9.0$ (Fig. 3.1.2) that a reaction center is forming at some distance from the contact discontinuity where the reactant is ~65 percent burnt. We emphasize that the width of the less burnt "finger" is about 3, i.e., it contains 25-30 computational zones and cannot be attributed to a numerical error. We will have a chance to see in the later simulations as well that the formation of a hot spot at some distance from the contact is a general fact. It can be given a rather simple explanation. In the presence of a spatial gradient in Z, the maximum of the reaction rate is attained

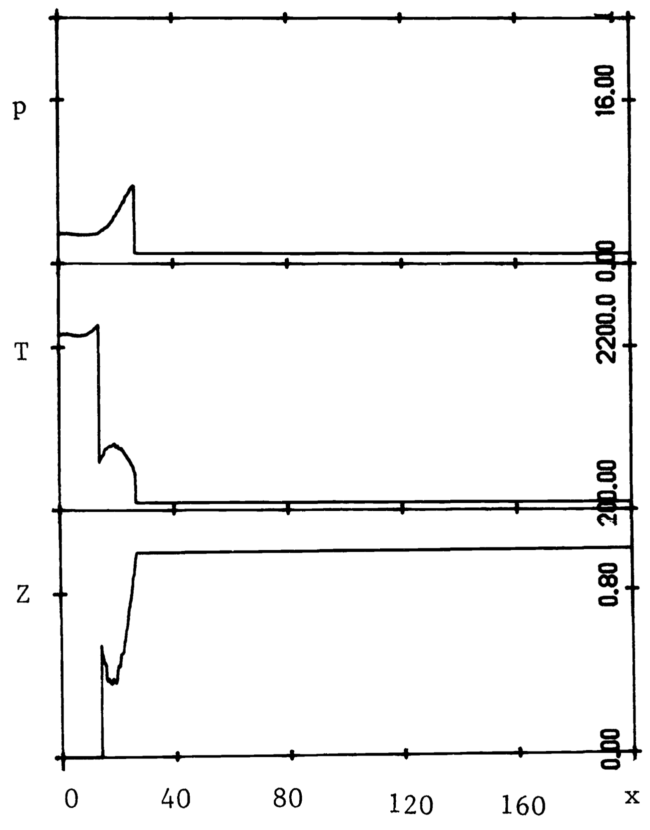


FIG. 3.1.2. *Pressure, temperature, and mass fraction distributions at* $t = 9.0$.

not at the contact but at some distance from it. This fact was overlooked by the previous asymptotic theories of homogeneous initiation that did not provide for spatial gradients in Z.

At $t = 15.0$ (Fig. 3.1.3), the fuel in the secondary reaction center is almost completely consumed. By this time there is already a fully developed reaction wave. It seems that the completion of reaction in the secondary hot spot accelerates the front pressure (see Fig. 3.1.1). After that time the spatial distribution of the state variables experience just minor quantitative change to develop into a full-fledged CJ detonation. We display a snapshot at $t = 48.0$ (Fig. 3.1.4).

The numerical resolution for the calculations discussed above was taken as 10 meshpoints per length unit. These calculations were a refinement of a coarse mesh calculation (five meshpoints per length unit) that produced a virtually identical time history.

3.2. Strong failure. For this simulation we take $E^+ = 15$. Of course, by the appropriate choice of initial data it is possible to demonstrate the strong failure for $E^+ = 10$ as well. We selected a higher activation energy for the present simulation to contrast it with the simulation of the next section. The temperature in the reaction center is taken with height six and width one.

After the initial pressure rise, the pressure history demonstrates a gradual pressure decay that is characteristic for a purely nonreactive shock. We display snapshots of spatial distribution of the state variables at three times: $t = 6, 26.9, 47.8$. The snapshots confirm that after initial burning the wave behaves like a nonreactive shock. The

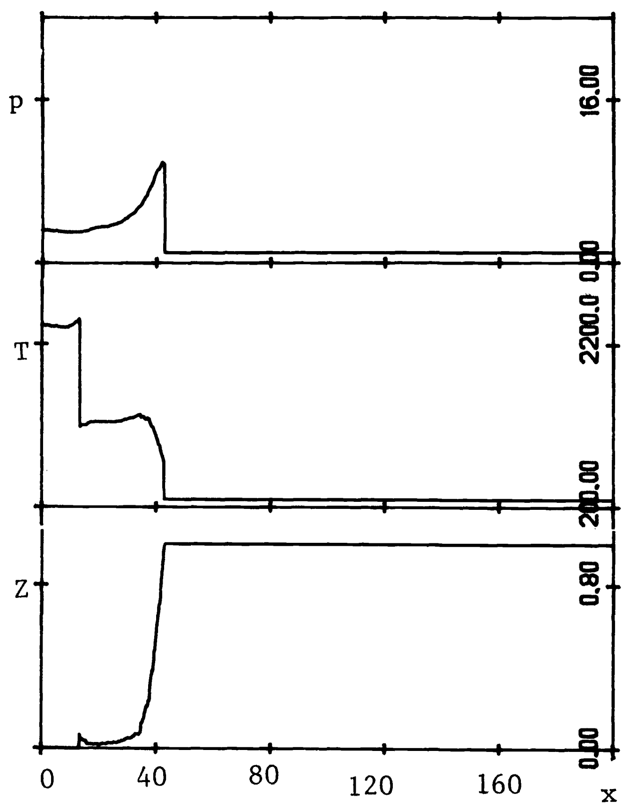


FIG. 3.1.3. *Pressure, temperature, and mass fraction distributions at* $t = 15.0$.

temperature rise caused by the strong initiating shock is cooled down by the rarefaction wave. The last frames give a clear picture of a nonreactive shock followed by rarefaction without any appreciable consumption of reactant.

The calculation was performed with resolution four meshpoints per unit length, which was a refinement of a coarser calculation with an identical time history.

3.3. Weak failure. This simulation is performed with the same initial and chemical data as the previous one except for the activation energy, which is lowered to $E^+ = 10$ (from $E^+ = 15$ in § 3.2). The pressure history for this case is almost identical to the one for strong failure and is not presented. The snapshots displayed in this section are taken at the same times as in § 3.2. They should be compared with Figs. 3.2.2–3.2.4. The snapshots demonstrate the effects of the lower activation energy, which yields less stiffness in the source terms. Thus, the reaction is appreciable in a broader range of temperature. For this reason the initial temperature rise produced by compression is sufficient to initiate the reaction (cf. the Z profile in Fig. 3.3.1). As the reaction proceeds almost 50 percent of the reactant is consumed at $t = 47$ (Fig. 3.3.3). Apparently the future evolution (beyond the computational time) would lead to formation of a secondary reaction center in the vicinity of the contact.

Numerical resolution for this simulation was four mesh zones per unit length (the same as for the strong failure case).

3.4. Weak initiation. For this simulation the chemical data are the same as before for the case of weak failure, but the initial pulse is taken to be more powerful. The

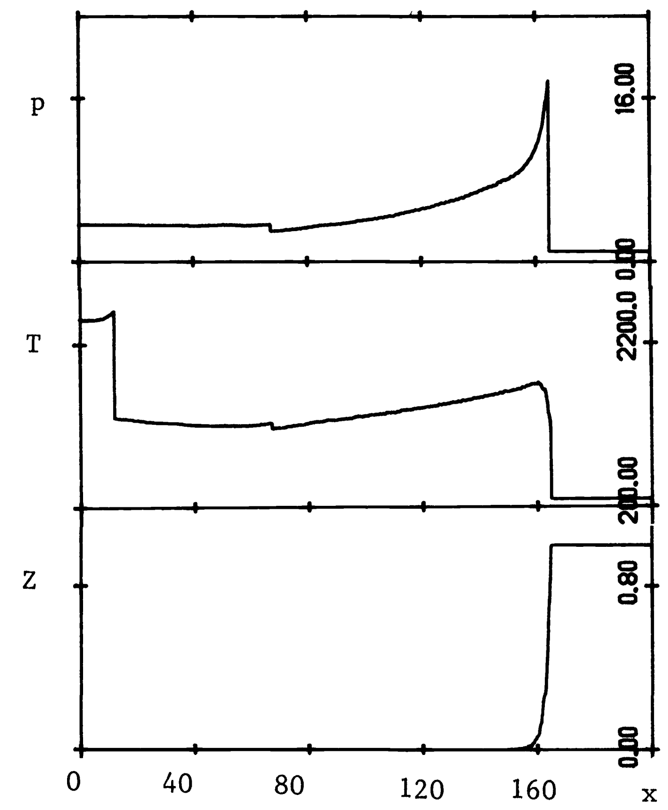


FIG. 3.1.4. *Pressure, temperature, and mass fraction distribution at* $t = 48.0$.

temperature is eight at the reaction center with width 2.5. The time evolution reveals an interesting sequence of events with rich structure.

At the first stage, until time ~25 the pressure history in Fig. 3.4.1 demonstrates a decrease in pressure that is characteristic for propagation and decay of an unsupported nonreactive shock. Comparison of the pressure and reactant spatial profiles shows that up to this time, the reaction wave and the precursor shock travel completely *separated from each other* by two to three length units (critical initiation in Lee's [3] terminology).

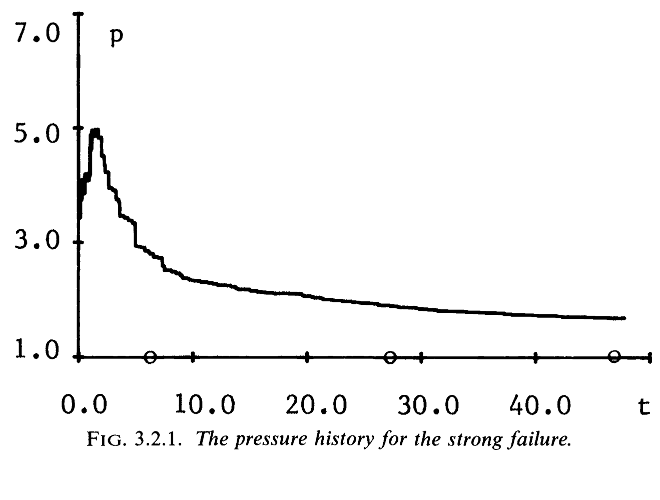


FIG. 3.2.1. *The pressure history for the strong failure.*

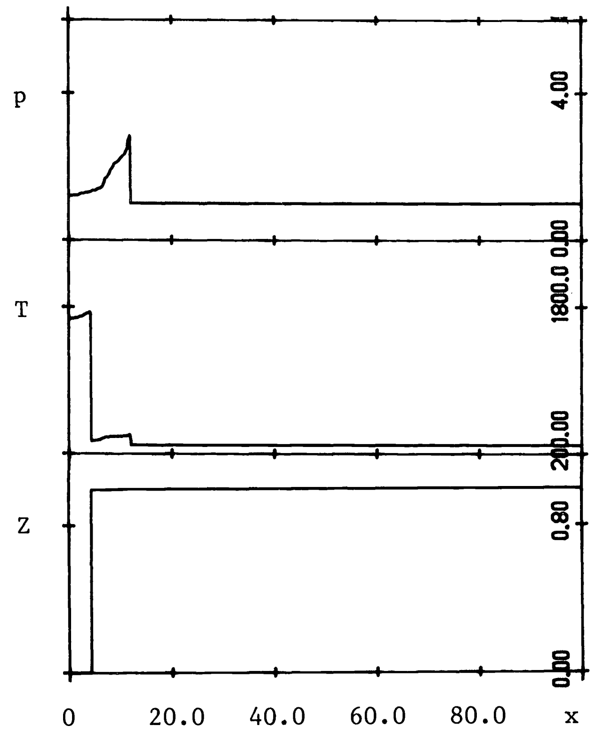


FIG. 3.2.2. *Pressure, temperature, and mass fraction distributions at* $t = 6.0$.

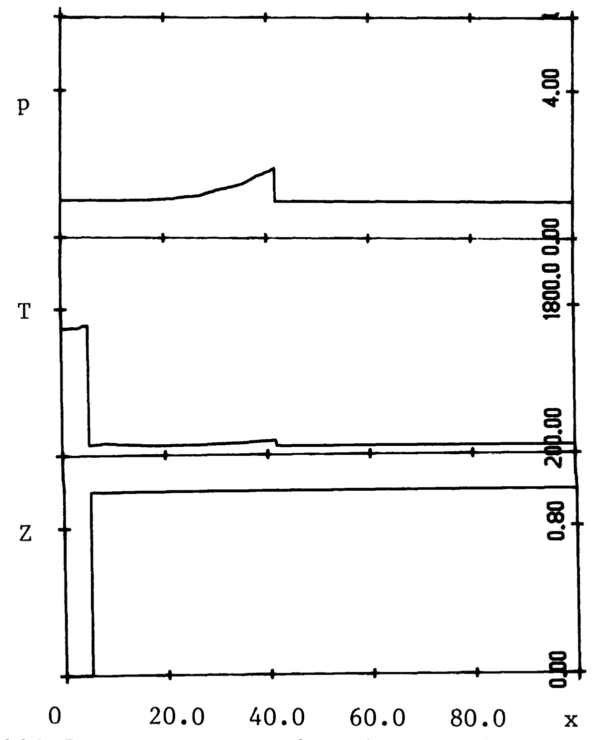


FIG. 3.2.3. *Pressure, temperature, and mass fraction distribution at* $t = 26.9$.

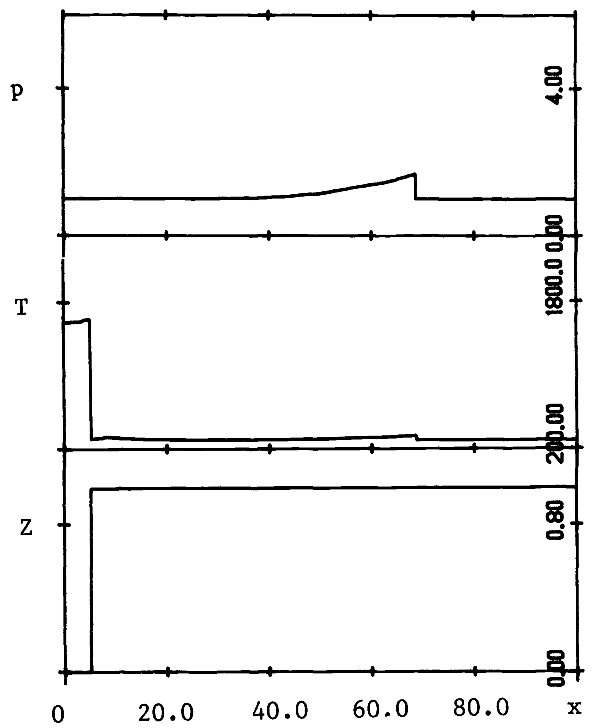


FIG. 3.2.4. *Pressure, temperature, and mass fraction distributions at* $t = 47.8$.

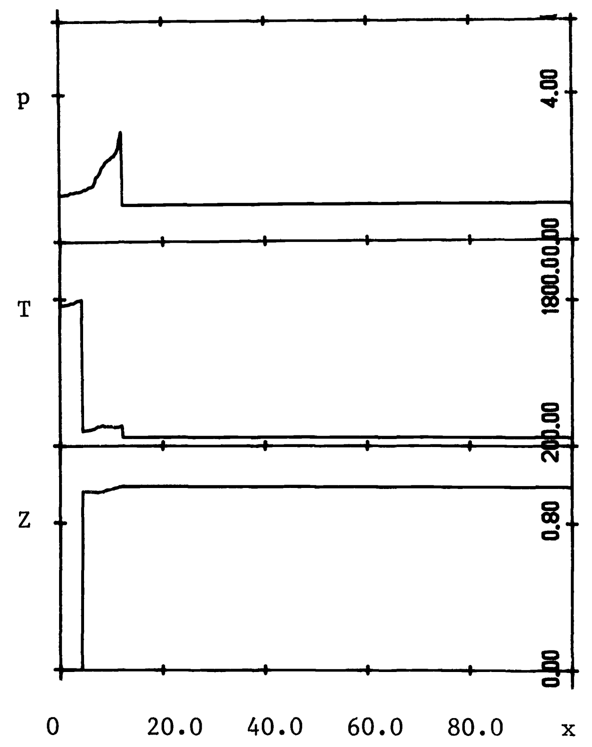


FIG. 3.3.1. *Weak failure: pressure, temperature, and mass fraction distributions at* $t = 6.0$.

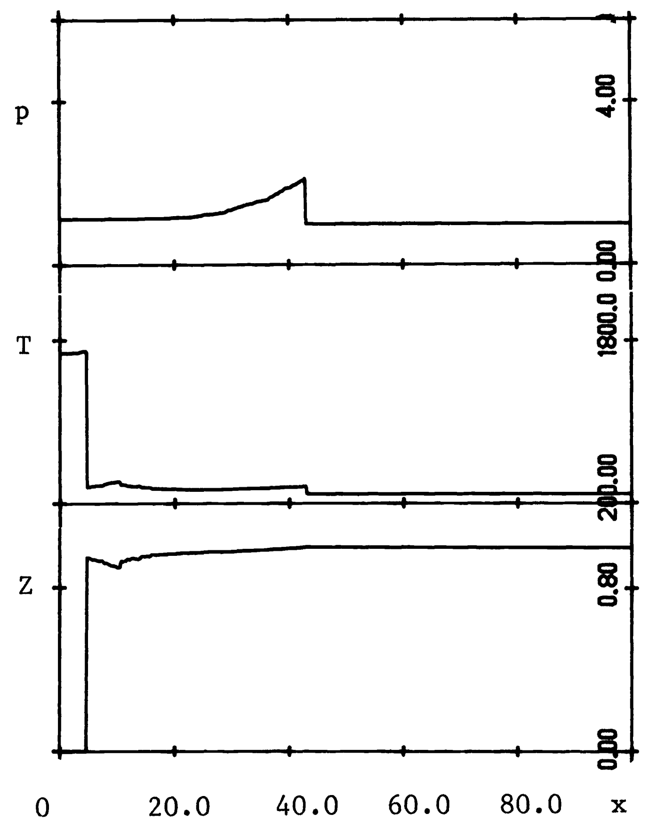


FIG. 3.3.2. *Weak failure: pressure, temperature, and mass fraction distributions at* $t = 26.9$.

For $25 < t < 30$ the reaction wave approaches the shock wave. This process causes a dramatic pressure increase. At $t \sim 30$ the reaction wave reaches the shock wave causing an even faster pressure increase up to the CJ value. Simultaneously the front accelerates dramatically.

The snapshot at $t = 10.0$ (Fig. 3.4.2) shows the beginning of the secondary reaction center formation in the vicinity of the contact similar to the one observed at the end of the weak failure simulation (Fig. 3.3.3). The secondary reaction center expands spatially (Fig. 3.4.3) and sends pressure waves in both directions. In this figure the pressure wave has just reflected at the origin. We note that the reactant profile is very smooth for values of Z close to 1. At $t = 30.0$ (Fig. 3.4.4), the reaction wave catches up with the precursor shock. The reaction zone has its characteristic form but still is rather wide. The temperature behind the reaction front is still relatively low. By time $t = 40$ (Fig. 3.4.5) the reaction zone narrows and the detonation wave takes a typical CJ profile. Numerical resolution for this simulation was eight zones per unit length.

3.5. Cylindrical geometry. In this simulation we illustrate a well-known experimental fact that the initiation process of a detonation wave can be quenched through geometric expansion effects. We took the data of the previous simulation (the weak initiation) and considered its evolution in cylindrical geometry.

The pressure history (Fig. 3.5.1) shows a fast decay of pressure at the front, much faster than for the planar case. In the first snapshot at $t = 2$ (Fig. 3.5.2) a very strong rarefaction due to geometric expansion is quite apparent. Its cooling effect (the temperature drop is ~30 percent) slows down burning so that even in the reaction

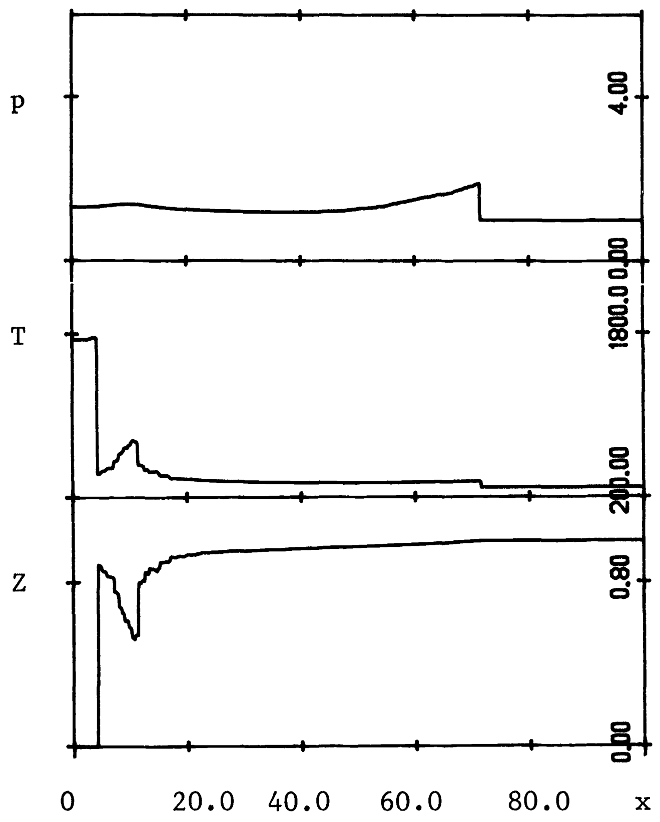


FIG. 3.3.3. *Weak failure: pressure, temperature, and mass fraction distributions at* $t = 47.8$.

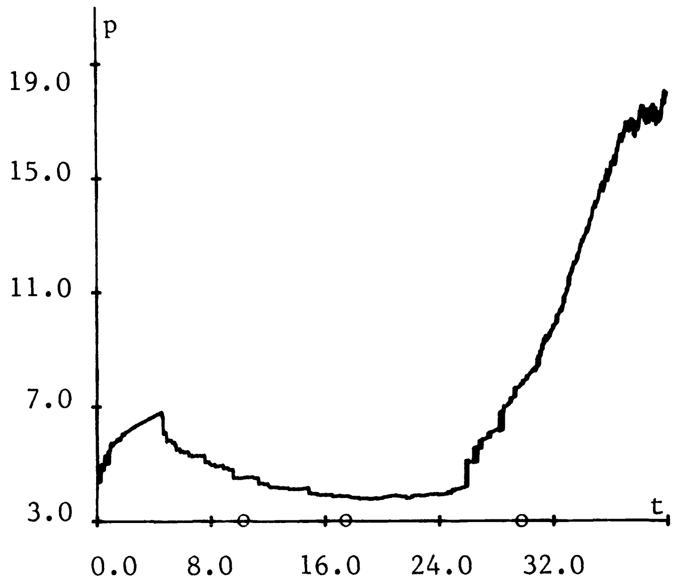


FIG. 3.4.1. *The pressure history for the weak initiation.*

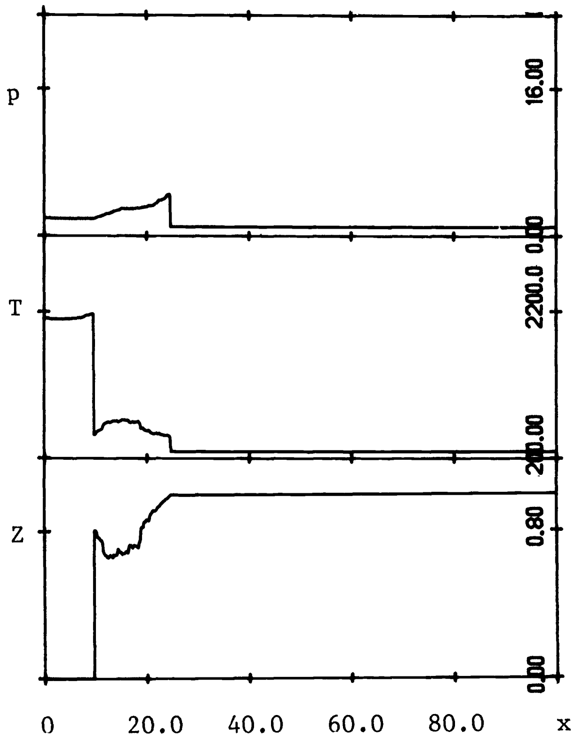


FIG. 3.4.2. *Pressure, temperature, and mass fraction distributions at* $t = 10.0$.

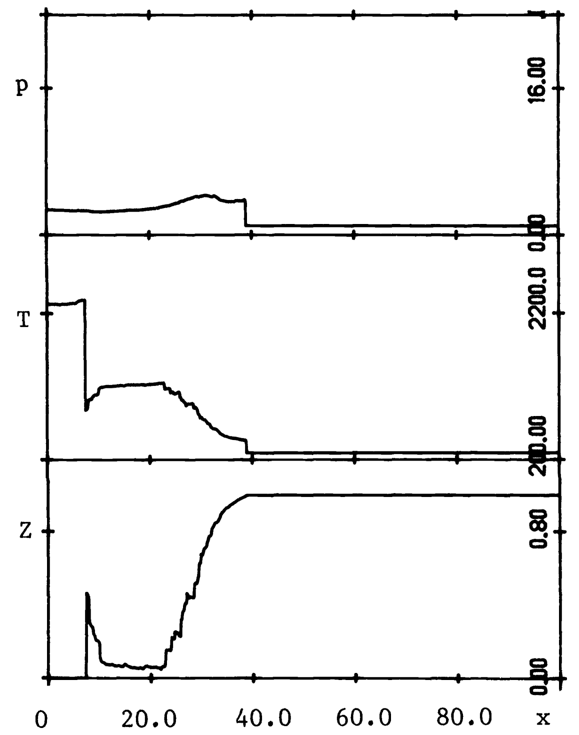


FIG. 3.4.3. *Pressure, temperature, and mass fraction distributions at* $t = 17.5$.

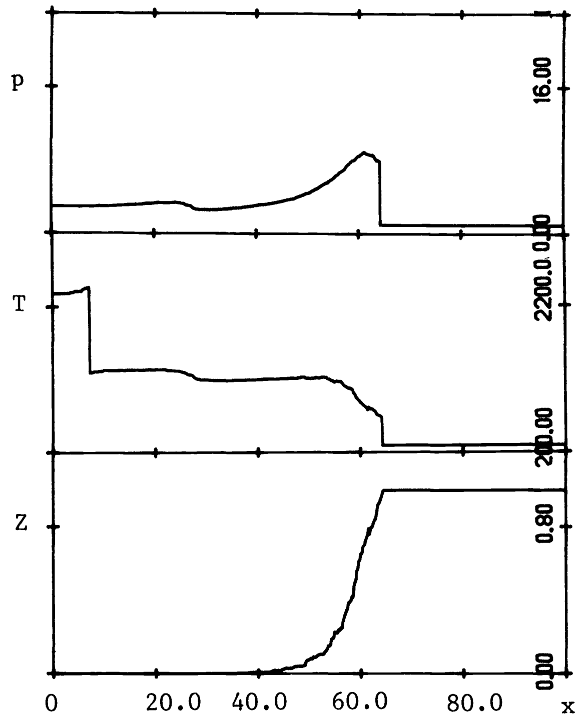


FIG. 3.4.4. *Pressure, temperature, and mass fraction distributions at* $t = 30.0$.

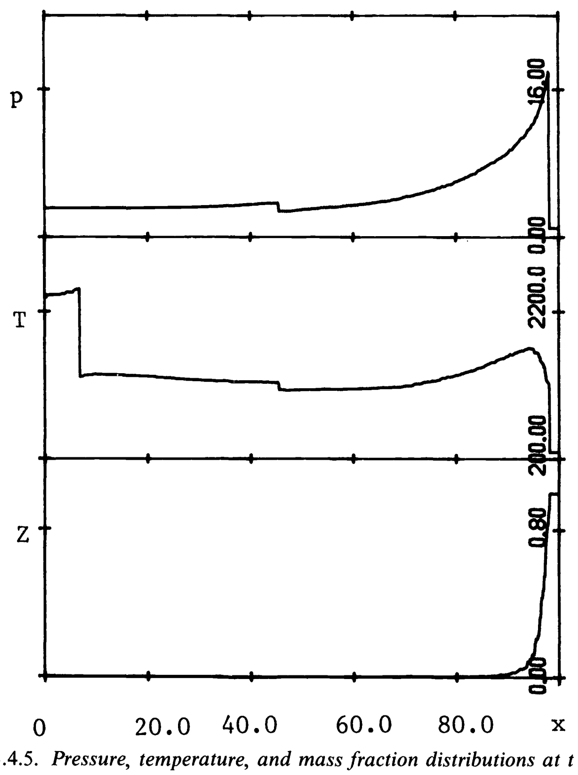


FIG. 3.4.5. *Pressure, temperature, and mass fraction distributions at* $t = 40.0$.

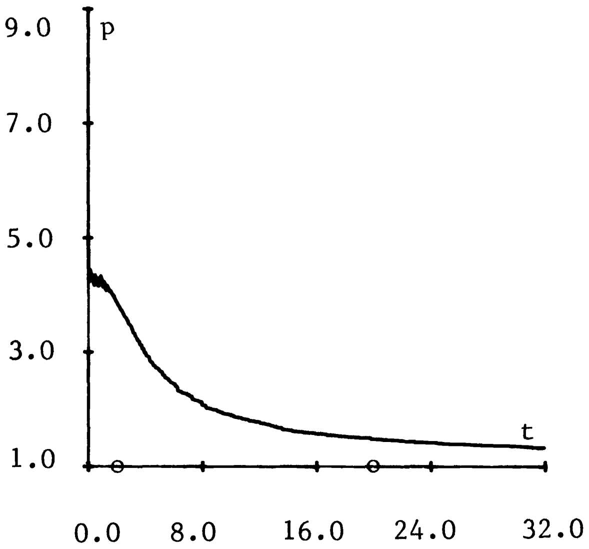


FIG. 3.5.1. *The pressure history for wave propagation in cylindrical geometry.*

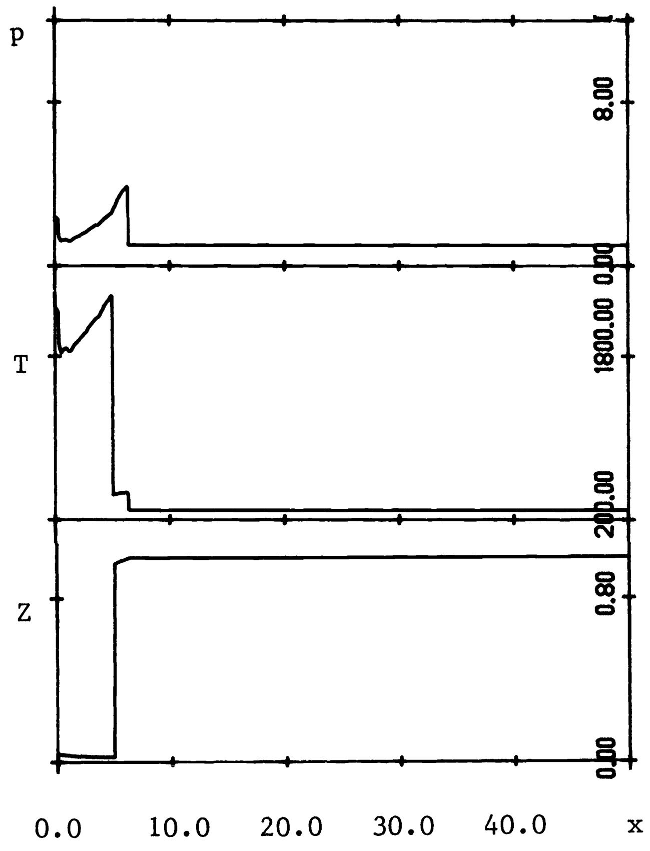


FIG. 3.5.2. *Pressure, temperature, and mass fraction distributions at* $t = 2.0$.

center the reactant is not fully consumed. In Fig. 3.5.3 we present one of the later profiles, $t = 20$. The pressure profile is typical for an unsupported unreactive shock. Hardly noticeable rippling shocks behind the leading shock are caused by the strong density variations in the beginning of the time evolution. The numerical resolution for this simulation was 16 meshpoints per unit length.

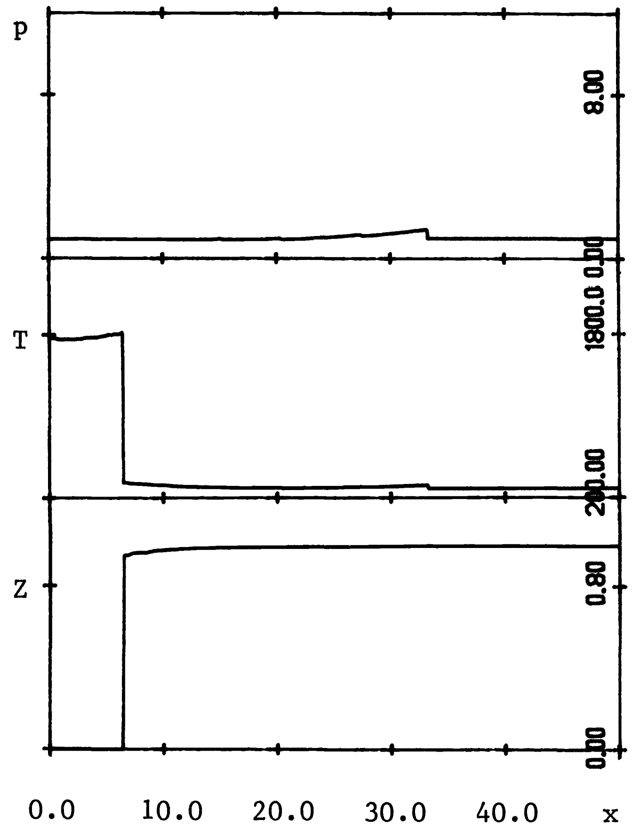


FIG. 3.5.3. *Pressure, temperature, and mass fraction profiles at* $t = 22.0$.

4. Resonant interaction of multiple reaction centers.

In the previous section we studied an important model situation where a single exothermic reaction center leads to formation of a secondary hot spot and to transition to detonation. It was observed experimentally (see, e.g., [23]) that transition to detonation is often characterized by the formation and interaction of several reaction centers. In this section we present a relevant numerical simulation. We also discuss a new mechanism of hot spot formation through high frequency wave interaction and resonant amplification. This mechanism should be particularly significant in condensed phases.

4.1. Initiation from two exothermic reaction centers.

The following route of transition to detonation was observed in experiments [1]. An exothermic reaction center is formed during initial stages of the transition. The center sends out a strong pressure wave and a reaction wave that are decoupled. The pressure wave in turn produces another reaction center that is eventually enhanced by the prior reaction wave. This enhancement leads to a stronger reaction wave and to eventual transition to detonation.

We simulate this route numerically with both reaction centers imbedded into the medium initially. Instead of being separated temporally, the reaction centers are thus separated spatially.

The chemical data for this simulation are the same as for the weak failure (§ 3.3), $q_0 = 10$, $E^+ = 10$, $\gamma = 1.4$. We consider two reaction centers situated at $0 \leqq x \leqq 1$ and $8 \leqq x \leqq 10$ with the temperature at the hot spots $T = 6$. The pulse at zero is the same as the pulse for the weak failure case. We remind the reader that because of the symmetry of the problem, the pulse of width one at the origin is just half the actual pulse (of width two). Strictly speaking, the initial data with two pulses for $x > 0$ represent three pulses of width two. The results of § 3.3 mean that neither of these reaction centers in isolation can initiate detonation.

The pressure history Fig. 4.1 demonstrates a pressure decrease in the precursor shock until $t \approx 20$ when the precursor is overtaken by a stronger wave. A similar event takes place at $t \approx 42$. At $t \approx 45$ the main reaction wave overtakes the leading shock. This causes a jump in pressure. After that the reaction intensifies explosively, leading to a full-fledged detonation with overshoot slightly above the CJ value. Closer to the end of the computational time we observe the onset of noisy low amplitude oscillations of the peak pressure that are characteristic of the random choice method.

More details of the evolution are revealed by the snapshots. In the first three figures for comparison we also present pressure distributions for the evolution in the nonreactive medium. We note that the pressure scales in these graphs are different because the nonreactive pressures are much lower. At $t = 6$ (Fig. 4.2), hot-spot formation has just started. Both pressure diagrams clearly show two leading shocks traveling to the right. These waves are the pressure waves sent in the positive direction by the initial reaction centers. In the reactive case the two leading shocks have a higher magnitude and move faster. The trailing waves come from the reflection at the origin and, in the reactive case, from the chemical reaction.

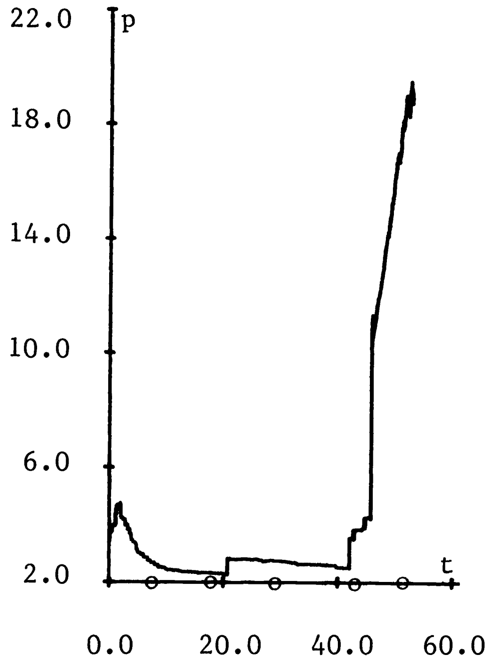


FIG. 4.1. *The pressure history for the initiation from two exothermic reaction centers.*

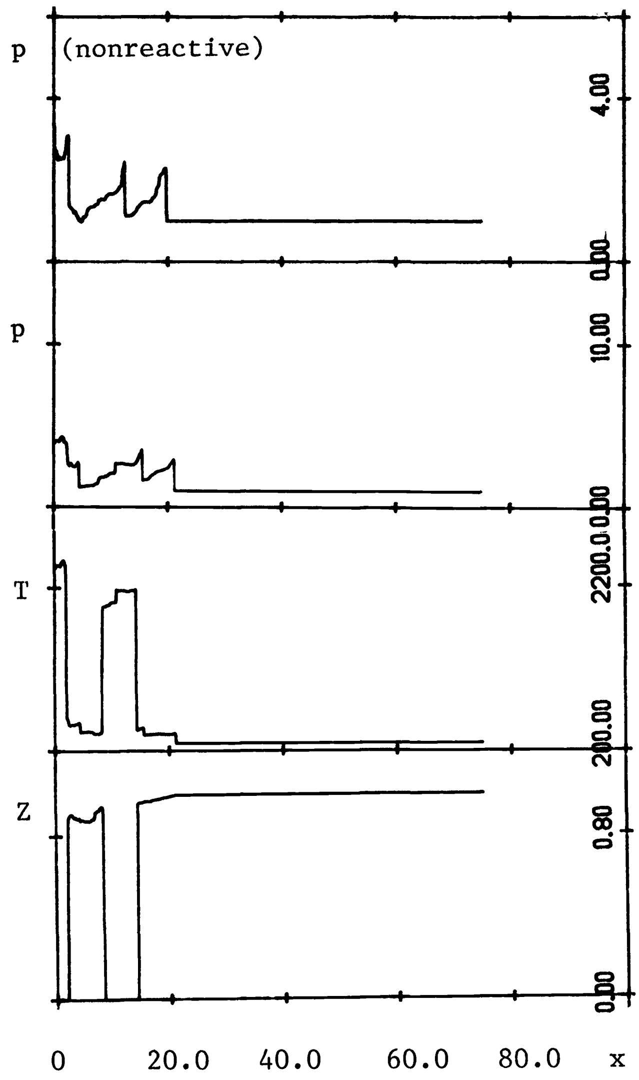


FIG. 4.2. *Pressure, temperature, and mass fraction distributions at $t = 6.0$. The top diagram represents the pressure distribution in the inert medium.*

In Fig. 4.3 ($t = 18.0$) the second leading shock of Fig. 4.2, which is being amplified by the reaction, is about to merge with the leading shock. This event will be the reason for the jump in the front pressure at $t \approx 20$ (Fig. 4.1). In contrast, the nonreactive shocks are well separated demonstrating a characteristic N-shape. The secondary hot spot between the two initial pulses is almost complete, whereas in the secondary hot spot ahead the reactant is about 40 percent consumed. We use an arrow to mark the pressure wave that will play an essential role in further development. This wave is being amplified by the rear hot spot. Then it will enter into resonance with the front hot spot to eventually overtake the front of the detonation wave. All the way through its development the front of this shock wave is positioned at the maximum of chemical reaction (which coincides with the point of maximal slope of Z).

At $t = 30.1$ (Fig. 4.4) formation of secondary hot spots is completed. There is a well-defined broad reaction zone. The arrow marked shock is still "sitting" on the

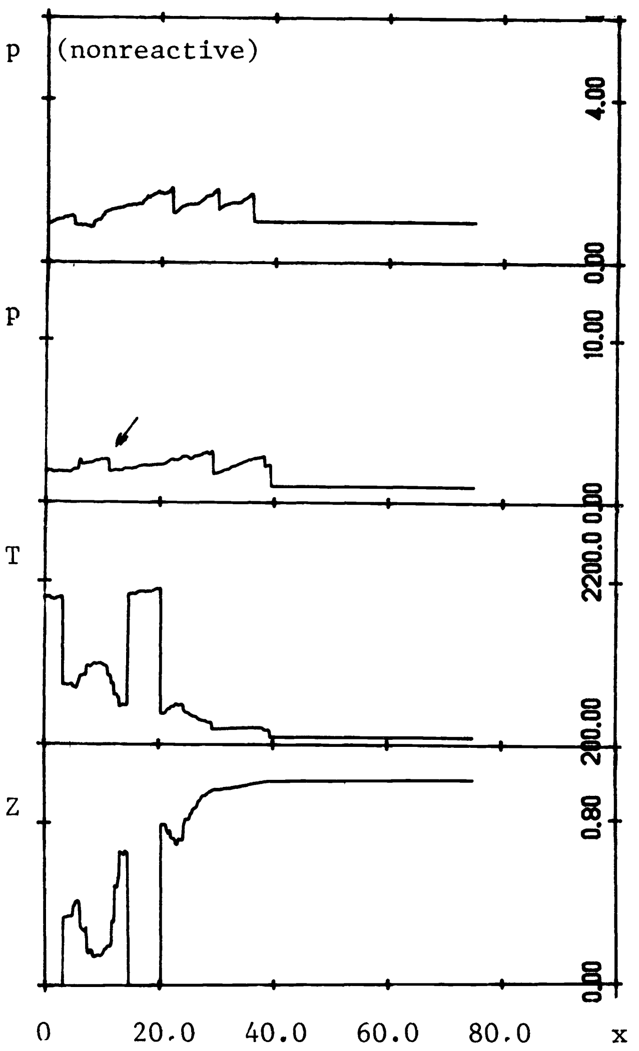


FIG. 4.3. *Pressure, temperature, and mass fraction distributions at* $t = 18.0$. *The top diagram represents the pressure distribution in the inert medium.*

maximum of the reaction rate. The nonreactive pressure diagram displays a series of N-waves.

In Figs. 4.5 and 4.6 the pressure magnitude is much higher and therefore the pressure is presented on a different scale. At time $t = 40.1$ (Fig. 4.5) the waves between the reaction wave and the precursor are about to overtake the leading shock. The reaction rate is steeper than before. At the front edge of the reaction zone the reaction rate is still quite smooth. The main reaction wave merges with the precursor wave at $t = 45$. This leads to a dramatic steepening of the reaction. Simultaneously the wave experiences a substantial acceleration. Finally, in Fig. 4.6 we display the wave pattern of a full fledged detonation.

The simulation in this section was performed with the numerical resolution of 16 meshpoints per unit length.

4.2. High frequency resonant enhancement of initiation. In this brief section we present results of simulations that support a mechanism of hot-spot formation that was recently suggested by Majda and Rosales [26] and recently developed in [7]. It

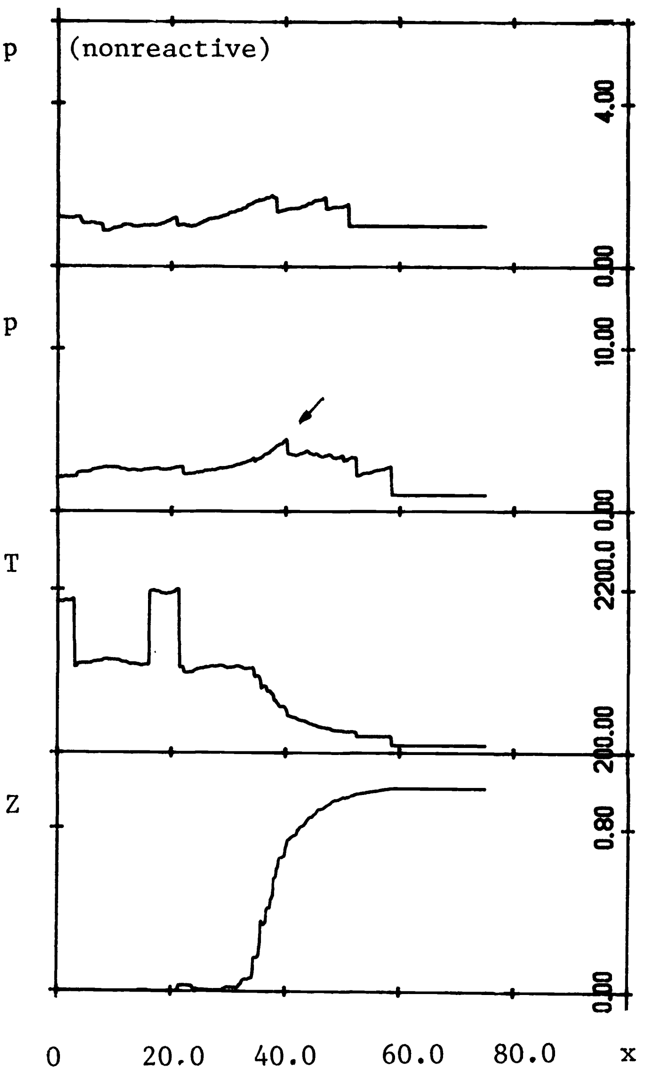


FIG. 4.4. *Pressure temperature, and mass fraction distributions at* $t = 30.1$. *The top diagram represents the pressure distribution in the inert medium.*

is shown in [26] and [7] that in the asymptotic limit of large activation energy, high frequency perturbations of the mean flow interact, enter into resonances, and enhance tremendously the initiation and formation of secondary hot spots. This mechanism should be especially relevant in the zone between the precursor shock and the flame. This zone, for real three-dimensional gases, is filled with high frequency acoustic noise. There is strong theoretical evidence (see [7]) that the mechanism of resonant enhancement should play an even more essential role for condense phase explosives. As is well known such explosives are crudely modeled by γ gas laws with large γ.

For the simulations in this section we used a space-time scaling different from the half-reaction scaling employed before. We normalize time by the homogeneous induction time with the corresponding change in the prefactor of the reaction rate:

$$w = \rho Z \exp\left(-\frac{E^+}{T}\right) \exp\,(E^+)/(\gamma - 1)q_0 E^+.$$

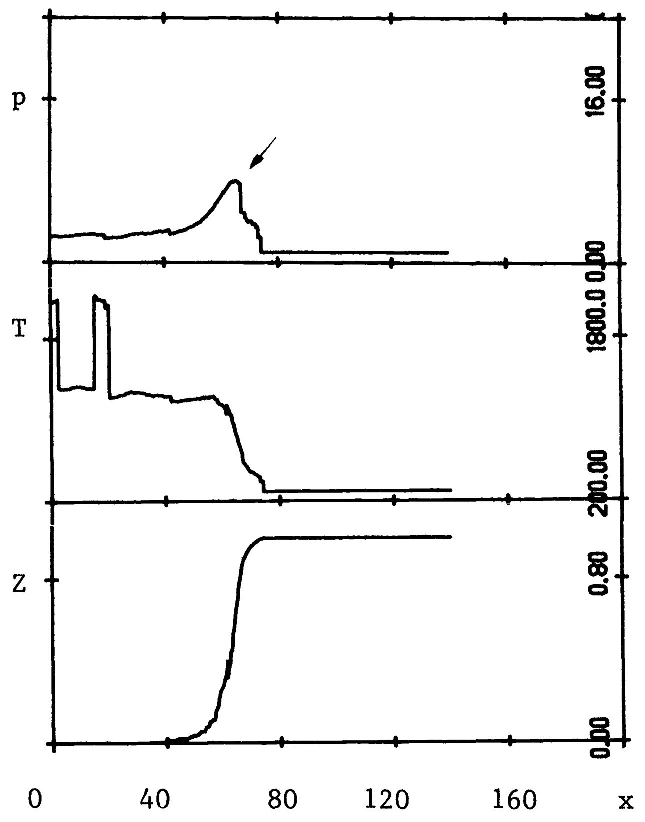


FIG. 4.5. *Pressure, temperature, and mass fraction distributions at* $t = 40.1$. *Note the scale change in pressure and temperature in comparison with Fig.* 4.4.

Thus, in these time units the induction time of a homogenous medium tends to one as E^+ tends to infinity. For finite values of E^+ the homogeneous induction time is somewhat larger than one.

We model high frequency perturbations by the square wave function depicted in Fig. 4.7. The scalings of the perturbation were suggested by [26]. We perturb the mean field $\rho \equiv \rho_0 = 1$, $p \equiv p_0 = 1$, $u \equiv u_0 = 0$, $Z \equiv Z_0 = 1$. The perturbed initial data are considered of the form

$$\rho = \rho_0 + \rho_0\phi, \quad p = p_0 + p_0\phi, \quad u = u_0 - \phi, \quad Z = Z_0. \tag{4.1}$$

Note that for the activation energies $E^+ = 20$, 50 used in our calculations below, these perturbations are extremely small, but still large enough to start thermal runaway. Without the wave interaction (say for a constant perturbation), this runaway takes time of order unity.

We did numerical simulations on the spatial interval $[0, 1/E^+]$ with periodic boundary conditions and with initial conditions (4.1). The numerical resolution for these simulations was 400 meshpoints in the computational domain. We do not include snapshots of the spatial distribution of space variables. It is sufficient to say that we can indicate two phases of wave evolution:

(i) A period of chemical-acoustic wave interaction without any substantial change in the magnitude of the state variables. It is natural to call this period an induction period.

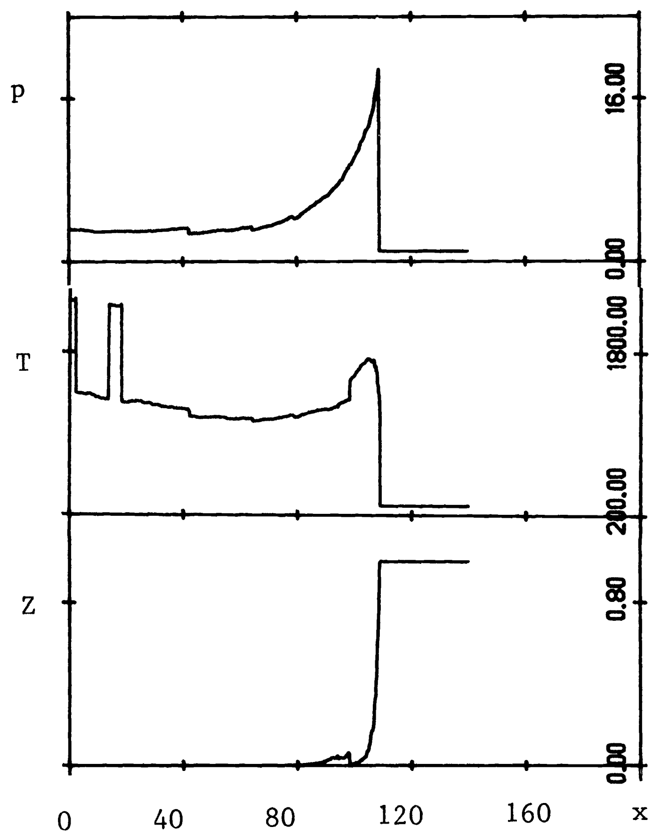


FIG. 4.6. *Pressure, temperature, and mass fraction distributions at* $t = 52.7$.

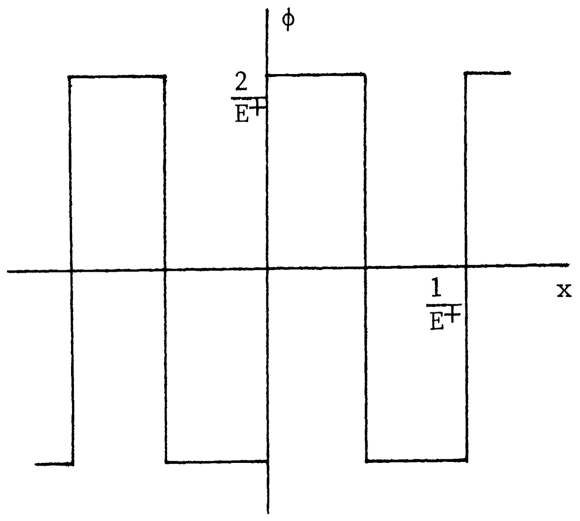


FIG. 4.7. *The high frequency square wave perturbation function.*

(ii) An explosive and very focused increase in magnitude that happens during times of two to three orders of magnitude shorter than the induction time. The explosion is characterized by the complete consumption of fuel in the focused interval where the explosion takes place. Within a few instants after the explosion, the reactant is entirely consumed everywhere.

We record the time until the explosion t_{ex} in Tables 1 and 2.

Tables 1 and 2 show the enhancement effect of the wave interaction: all the explosion times are less than one. But for the case with $\gamma = 3$ more closely modeling condense state explosives, this effect is much more dramatic: the explosion times are almost two orders of magnitude less than one. This agrees with the predictions from [7].

TABLE 1
($\gamma = 1.4$)

E^+	20	20	20	50	50	50
q_0	1	5	20	1	5	20
t_{ex}	0.92	0.70	0.67	0.654	0.625	0.621

TABLE 2
($\gamma = 3$)

E^+	20	20	50	50
q_0	1	20	1	20
t_{ex}	0.065	0.065	0.0275	0.0275

The results in Table 1 also demonstrate the tendency of explosion acceleration with increase in the heat release q_0. We believe that the same is true for $\gamma = 3$. Although the relative importance of this effect for $\gamma = 3$ is so small that it will require much more resolved calculations to verify it. Finally, in both cases the activation energy increase leads to earlier explosion. This speedup is more pronounced for $\gamma = 3$. It probably means that the large activation energy asymptotic limit is approached faster for $\gamma = 1.4$ than for $\gamma = 3$.

REFERENCES

[1] A. K. OPPENHEIM AND R. I. SOLOUKHIN, *Experiments in gasdynamics of explosions*, Ann. Rev. Fluid Mech., 5 (1973), pp. 531-558.
[2] J. H. S. LEE AND I. O. MOEN, *The mechanism of transition from deflagration to detonation in vapor cloud explosion*, Progr. Energy Combust. Sci., 6 (1980), pp. 359-389.
[3] J. H. S. LEE, *Initiation of gaseous detonation*, Ann. Rev. Phys. Chem., 28 (1977), pp. 75-104.
[4] W. FICKETT AND W. C. DAVIS, *Detonation*, University of California Press, Berkeley, CA, 1979.
[5] L. J. ZAJAC AND A. K. OPPENHEIM, *Dynamics of an explosive reaction center*, AIAA J., 9 (1971), pp. 545-553.
[6] J. M. MEYER AND A. K. OPPENHEIM, *Dynamic response of a plane-symmetrical exothermic reaction center*, AIAA J., 10 (1972), pp. 1509-1533.
[7] R. ALMGREN, A. MAJDA, AND R. ROSALES, *Rapid initiation in condensed phases through resonant nonlinear acoustics*, Phys. Fluids, to appear.
[8] C. L. MADER, *Numerical Modeling of Detonations*, University of California Press, Berkeley, CA, 1979.
[9] W. FICKETT AND W. W. WOODS, *Flow calculation for pulsating one-dimensional detonation*, Phys. Fluids, 9 (1966), pp. 903-916.
[10] G. E. ABOUSEIF AND T. Y. TOONG, *Theory of unstable one-dimensional detonations*, Combust. and Flame, 45 (1982), pp. 67-94.

[11] E. S. ORAN, T. R. YOUNG, J. P. BORIS, AND A. COHEN, *Weak and strong ignition* I, Combust. and Flame, 48 (1982), pp. 135-148.

[12] E. S. ORAN AND J. P. BORIS, *Weak and strong ignition* II, Combust. and Flame, 48 (1982), pp. 149-161.

[13] K. KAILASANATH AND E. S. ORAN, *Ignition of flamelets behind incident shock waves and the transition to detonation*, Combust. Sci. Tech., 34 (1983), pp. 345-362.

[14] 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.

[15] A. BOURLIOUX, *Analysis of numerical methods in a simplified detonation model*, in preparation.

[16] A. A. BORISOV, *On the origin of exothermic centers in gaseous mixtures*, Acta Astronautica, 1 (1974), pp. 909-920.

[17] A. MAJDA AND V. ROYTBURD, *Numerical modeling of the initiation of reacting shock waves*, Computational Fluid Mechanics and Reacting Gas Flows, B. Engquist, M. Luskin, and A. Majda, eds., The IMA Volumes in Mathematics and its Applications, Vol. 12, Springer-Verlag, New York, 1988, pp. 195-217.

[18] B. BUKIET, *The effect of curvature on detonation speed*, SIAM J. Appl. Math., 49 (1989), pp. 1433-1446.

[19] A. CHORIN, *Random choice methods with applications for reacting gas flow*, J. Comput. Phys., 25 (1977), pp. 253-272.

[20] I.-L. CHERN, J. GLIMM, O. MCBRYAN, B. PLOHR, AND S. YANIV, *Front tracking for gas dynamics*, J. Comput. Phys., 62 (1983), pp. 83-110.

[21] I.-L. CHERN AND P. COLELLA, *A conservative front tracking method for hyperbolic conservation laws*, Lawrence Livermore National Laboratory, Preprint UCRL 97196, 1987; J. Comput. Phys., to appear.

[22] A. BOURLIOUX, A. MAJDA, AND V. ROYTBURD, *Theoretical and numerical structure for unstable one-dimensional detonations*, SIAM J. Appl. Math., submitted.

[23] P. A. URTIEU AND A. K. OPPENHEIM, *Experimental observations of the transition to detonation in an explosive gas*, Proc. Roy. Soc. London Ser. A, 295 (1966), pp. 13-28.

[24] I. O. MOEN, J. W. FUNK, S. A. WARD, G. M. RUDE, AND P. A. THIBAULT, *Detonation length scales for fuel-air explosives*, Progr. Aeronautics Astronautics, 94 (1985), pp. 55-79.

[25] A. KAPILA AND V. ROYTBURD, *Transition to detonation: A numerical study*, Numerical Combustion, A. Dervieux and B. Larrouturou, eds., Lecture Notes in Physics 351, Springer-Verlag, Berlin, 1989, pp. 375-382.

[26] A. MAJDA AND R. R. ROSALES, *Nonlinear mean field-high frequency wave interactions in the induction zone*, SIAM J. Appl. Math., 47 (1987), pp. 1017-1039.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 5, pp. 975-989, September 1990

COLUMN RELAXATION METHODS FOR LEAST NORM PROBLEMS*

ACHIYA DAX† AND BRIAN BERKOWITZ†

Abstract. The paper derives column relaxation schemes for calculating the ℓ_p solution of an inconsistent system of linear equations. The need for such methods arises when the system to be solved is large, sparse, and unstructured. Special attention is given to each of the cases $p=1$, $1<p<2$, $2<p<\infty$, and $p=\infty$. Numerical experiments illustrate the ability of the proposed schemes to handle large ℓ_1 and ℓ_∞ problems.

Key words. column relaxation, ℓ_1 and ℓ_∞ solutions of inconsistent systems of linear equations, large unstructured systems

AMS(MOS) subject classifications. 65K99, 65O99

1. Introduction.

This paper presents efficient relaxation methods for solving the least norm problem

$$\text{(1.1)} \qquad \text{minimize } F(\mathbf{x}) = \|A\mathbf{x}-\mathbf{b}\|_p$$

where A is a real $m\times n$ matrix, $\mathbf{b}=(b_1,\cdots,b_m)^T\in\mathbb{R}^m$, and $\mathbf{x}=(x_1,\cdots,x_n)^T\in\mathbb{R}^n$ denotes the vector of unknowns. The norm that defines our objective function is the ℓ_p norm: Given a point $\mathbf{y}=(y_1,\cdots,y_m)^T\in\mathbb{R}^m$ then

$$\|y\|_p=\left(\sum_{i=1}^m |y_i|^p\right)^{1/p} \quad \text{when } 1\leqq p<\infty,$$

$$\|y\|_\infty=\max_i |y_i| \quad \text{when } p=\infty.$$

It is assumed here that the linear system $A\mathbf{x}=\mathbf{b}$ is inconsistent and that A is large, sparse, and unstructured. In such a case we usually store the nonzero elements of A either in a row after row order, or in a column after column order. The first mode is called "row-storage," whereas the second mode is called "column-storage." Consequently, it is convenient to solve the problem either by a "row-relaxation" method or by a "column-relaxation" method. The basic iteration of a row-relaxation scheme is composed of one sweep along the rows of A. Similarly, the basic iteration of a column-relaxation scheme is composed of one sweep along the columns of A.

Row-relaxation methods have important applications in the field of image reconstruction from projections. In this field, it is possible to avoid storing A; instead, the nonzero entries of the ith row are generated from the experimental data each time they are needed. Therefore, these methods are sometimes called "row-generation" or "row-action" methods. The reader is referred to Censor [4] for an excellent survey of row-relaxation methods and the special environment that characterizes their use. For a detailed convergence analysis of these methods see, e.g., Björck and Elfving [2], Elfving [11], or Dax [9].

The extension of row-relaxation methods to solve least norm problems with $p\neq 2$ is studied elsewhere (see Dax [10]). This paper concentrates on column-relaxation methods. The basic iteration of such a method is composed of n steps. At the jth step, $j=1, 2,\cdots, n$, x_j alone is changed in an attempt to reduce the objective function

* Received by the editors June 15, 1988; accepted for publication (in revised form) August 11, 1989.

† Hydrological Service, P.O. Box 6381, Jerusalem 91063, Israel.

value, whereas all other variables are kept fixed. We will use $\mathbf{x}^{k,j} \in \mathbb{R}^n$ to denote the current estimate at the beginning of the jth step of the kth iteration, $k = 1, 2, \cdots$, and

$$\mathbf{r}^{k,j} = A\mathbf{x}^{k,j} - \mathbf{b} \tag{1.2}$$

to denote the corresponding residual vector. The jth column of A is denoted by $\mathbf{c}_j$, whereas $\mathbf{e}_j$ denotes the jth column of the $n \times n$ unit matrix. With this notation, $\mathbf{x}^{k,j+1}$ is obtained from $\mathbf{x}^{k,j}$ by the rule

$$\mathbf{x}^{k,j+1} = \mathbf{x}^{k,j} + \lambda^{k,j}\mathbf{e}_j \tag{1.3}$$

where $\lambda^{k,j}$ is computed by a "steplength" algorithm that is aimed at minimizing the one-parameter function

$$f(\lambda) = \|A(\mathbf{x}^{k,j} + \lambda\mathbf{e}_j) - \mathbf{b}\|_p^p = \|\lambda\mathbf{c}_j + \mathbf{r}^{k,j}\|_p^p. \tag{1.4}$$

The details of the one-dimensional search are discussed in the next sections. Note that $\lambda^{k,j}$ is not necessarily the exact minimizer of $f(\lambda)$. For $p = 2$ the minimizer of $f(\lambda)$ lies at the point

$$\hat{\lambda} = -\frac{\mathbf{c}_j^T\mathbf{r}^{k,j}}{\mathbf{c}_j^T\mathbf{c}_j} \tag{1.5}$$

and setting

$$\lambda^{k,j} = \hat{\lambda} \tag{1.6}$$

coincides with the Gauss-Seidel relaxation method for solving the normal equations

$$A^TA\mathbf{x} = A^T\mathbf{b}. \tag{1.7}$$

Similarly, the choice

$$\lambda^{k,j} = \omega\hat{\lambda}, \tag{1.8}$$

where $0 < \omega < 2$ is a preassigned relaxation parameter, yields the successive over-relaxation (SOR) method for solving (1.7). Yet it is well known that a "tuned" SOR scheme may converge much faster than the Gauss-Seidel scheme.

For $2 < p < \infty$ the analogue of (1.8) is

$$\lambda^{k,j} = -\frac{\omega f'(0)}{f''(0)}, \tag{1.9}$$

which can be viewed as a special case of the nonlinear SOR method (see Ortega and Rheinboldt [15]). If, however, the starting point lies far from a minimizer, the above scheme is likely to diverge. A simple remedy that solves this difficulty is to shorten the steplength until it satisfies

$$f'(\lambda^{k,j}) \leqq 0 \tag{1.10}$$

(see the next section).

The situation is more complicated when $1 < p < 2$. In this case, the second derivatives of $F(\mathbf{x})$ are not defined at points where the residual vector $A\mathbf{x} - \mathbf{b}$ has zero components. Hence, the basic steplength (1.9) must be modified whenever $f''(0)$ is undefined or too large. Of course, there are many possible ways to modify the line search. However, since $|f''(\lambda)|$ is not bounded, it is difficult to show that the resulting sequence $\{\mathbf{x}^{k,j}\}$ approaches a minimizer of $F(\mathbf{x})$. For example, if $\lambda^{k,j}$ is defined as the exact minimizer of $f(\lambda)$, then the resulting method is sometimes called "the method that changes one variable at a time." Yet it has been shown by Powell [16] that this method may fail to minimize a continuously differentiable function. The solution proposed in this paper is to replace $F(\mathbf{x})$ with a hyperbolic approximating function (a HAP function) of the form

$$H(\mathbf{x})=\left\{\sum_{i=1}^{m}[(\mathbf{a}_i^T\mathbf{x}-b_i)^2+\varepsilon^2]^{p/2}\right\}^{1/p} \tag{1.11}$$

where $\mathbf{a}_i^T$ denotes the ith row of A and ε is a small positive constant. The HAP function (1.11) is a convex twice continuously differentiable function that satisfies

$$|F(\mathbf{x})-H(\mathbf{x})|\leqq m^{1/p}\varepsilon \tag{1.12}$$

for all $\mathbf{x}\in\mathbb{R}^n$ (see § 3). Hence there is no difficulty in applying the modified SOR method to minimize $H(\mathbf{x})$.

On the other hand, as ε approaches zero, the problem of minimizing $H(\mathbf{x})$ may become ill-conditioned. Thus in practice, we usually start with a relatively large value of ε and gradually reduce it. Another related question, as yet unsolved, is whether a minimizer of $H(\mathbf{x})$ is in fact near to a minimizer of $F(\mathbf{x})$. The idea of replacing a nonsmooth objective function by a HAP function is borrowed from the field of multifacility location problems (see Eyster, White, and Wierwille [12], or Francis and White [14]).

If $p=1$ or $p=\infty$, the objective function (1.1) is a polyhedral convex function. In this case, the one-parameter function that is minimized at the jth step is redefined as

$$f(\lambda)=\|\lambda\mathbf{c}_j+\mathbf{r}^{k,j}\|_p. \tag{1.13}$$

In other words, here $f(\lambda)$ is a convex piecewise linear function whose minimizer lies at one of its "breakpoints." Furthermore, the sparseness of A implies that $\mathbf{c}_j$ is likely to have only a few nonzero elements. Hence the number of breakpoints is expected to be small and a minimizer of $f(\lambda)$ is easily computed. This suggests that here $\lambda^{k,j}$ should be defined as a minimizer of $f(\lambda)$. The disadvantage of this approach is that the sequence $\{\mathbf{x}^{k,j}\}$ may converge to a nonoptimal point, and it is therefore necessary to take some precaution against this possibility (see §§ 4 and 5).

The reader who is familiar with minimization techniques of "small" problems may consider the column relaxation method as a variation of the method that changes one variable at a time. This latter method has a notorious reputation for being inefficient and unreliable (see Fletcher [13, p. 15]). It is therefore tempting to dismiss the proposed method on the same grounds. However, for large sparse problems, the situation is quite different, and the method enjoys many advantages. The main advantage is that the implementation of the method is rather simple. Let us take, for example, the case $2<p<\infty$. Let N_j denote the subset of $\{1,2,\cdots,m\}$ that contains the indices of the nonzero components of $\mathbf{c}_j$, and let α_i and β_i, $i=1,\cdots,m$, denote the components of $\mathbf{c}_j$ and $\mathbf{r}^{k,j}$, respectively. Then (1.4) can be rewritten in the form

$$f(\lambda)=\sum_{i\in N_j}|\lambda\alpha_i+\beta_i|^p \tag{1.14}$$

while the basic steplength (1.9) satisfies

$$\lambda^{k,j}=-\omega\left(\sum_{i\in N_j}w_i\alpha_i\beta_i\right)\Big/\left[(p-1)\left(\sum_{i\in N_j}w_i\alpha_i^2\right)\right] \tag{1.15}$$

where

$$w_i=|\beta_i|^{p-2}. \tag{1.16}$$

Consequently, since the only components of the residual vector that must be updated by the change in x_j are those whose indices belong to N_j, both the computational effort per iteration and the storage requirements are small. Also, as with row-generation methods, it is possible in certain applications to avoid storing A, and instead generate anew the nonzero entries of $\mathbf{c}_j$ each time (see Examples 2 and 3 of § 7).

Finally, we note that another way to overcome difficulties that arise when $p<2$ is to replace (1.16) with

$$w_i = (\max\{|\beta_i|, \varepsilon\})^{p-2} \tag{1.17}$$

where ε is a small positive constant. This idea is investigated in [10].

2. Column relaxation for $2<p<\infty$. In this case, $f(\lambda)$ is twice continuously differentiable:

$$f'(0)=p \sum_{i\in N_j} w_i\alpha_i\beta_i, \tag{2.1}$$

$$f''(0)=p(p-1) \sum_{i\in N_j} w_i\alpha_i^2. \tag{2.2}$$

Consequently, $\hat{\lambda}=-f'(0)/f''(0)$ is the steplength of Newton's method for minimizing $f(\lambda)$, whereas $\tilde{\lambda}=\omega\hat{\lambda}$ is the steplength of the nonlinear SOR scheme. However, as this scheme is based on a second-order Taylor expansion of a highly nonlinear function, the nonlinear SOR method is likely to diverge whenever the starting point lies far from a minimizer. To avoid this possibility, we introduce the following modification. Define $\rho_\ell=(1/2)^\ell\tilde{\lambda}$, $\ell=0, 1, 2, \cdots$, and set $\lambda^{k,j}$ to be the first number in the sequence $\{\rho_\ell\}$ that satisfies

$$f'(\rho_\ell)\leqq 0. \tag{2.3}$$

Of course, if N_j is empty, or $f'(0)=0$, then $\lambda^{k,j}$ is set to zero.

THEOREM 1. *The above column relaxation method is well defined. The sequence* $\{F(\mathbf{x}^{k,j})\}$ *is monotone decreasing,* $\lim_{k\to\infty}|\lambda^{k,j}|=0$ *for* $j=1,\cdots,n$, *and any cluster point of the sequence* $\{\mathbf{x}^{k,j}\}$, *if it exists, solves* (1.1).

Proof. The special structure of $f(\lambda)$ implies that it has a unique minimizer, say λ^*. Assume for simplicity that $f'(0)<0$. Then $\lambda^*>0$ and $f'(\lambda)\leqq 0$ for all $0\leqq\lambda\leqq\lambda^*$. These properties ensure that the bisection process terminates in a finite number of steps at a point that satisfies

$$f(0)>f(\lambda^{k,j}). \tag{2.4}$$

Hence the relation

$$(F(\mathbf{x}^{k,j}))^p-(F(\mathbf{x}^{k,j+1}))^p=f(0)-f(\lambda^{k,j}) \tag{2.5}$$

indicates that the sequences $\{(F(\mathbf{x}^{k,j}))^p\}$ and $\{F(\mathbf{x}^{k,j})\}$ are monotone decreasing. Therefore, since these sequences are bounded below by zero, the difference (2.5) tends to zero as k increases. Also, $f''(\lambda)$ satisfies the inequalities

$$\begin{aligned}0\leqq f''(\lambda)&=p(p-1)\sum_{i\in N_j}\alpha_i^2|\lambda\alpha_i+\beta_i|^{p-2}\\ &\leqq p(p-1)\alpha\sum_{i\in N_j}|\lambda\alpha_i+\beta_i|^{p-2}\\ &\leqq p(p-1)\alpha\nu\{\max_{i\in N_j}|\lambda\alpha_i+\beta_i|\}^{p-2}\\ &\leqq p(p-1)\alpha\nu\left\{\left(\sum_{i\in N_j}|\lambda\alpha_i+\beta_i|^p\right)^{1/p}\right\}^{p-2}\\ &\leqq p(p-1)\alpha\nu(f(\lambda))^{(p-2)/p}\end{aligned} \tag{2.6}$$

where $\alpha=\max\{\alpha_i^2 \mid i\in N_j\}$ and ν denotes the number of nonzero elements in $\mathbf{c}_j$ (i.e., ν is the number of elements in N_j). Furthermore, since $f(\lambda)$ is monotone decreasing in the interval $[0, \lambda^*]$,

$$0\leqq f''(\lambda)\leqq p(p-1)\alpha\nu(f(0))^{(p-2)/p}. \tag{2.7}$$

Also, since $(f(0))^{1/p}$ cannot exceed $F(\mathbf{x}^{1,1})$, the constant

$$\gamma=p(p-1)\alpha\nu(F(\mathbf{x}^{1,1}))^{p-2} \tag{2.8}$$

satisfies

$$f''(\lambda)\leqq\gamma \tag{2.9}$$

for all $0\leqq\lambda\leqq\lambda^*$. The last bound implies that the quadratic function

$$\phi(\lambda)=f(0)+\lambda f'(0)+(1/2)\gamma\lambda^2 \tag{2.10}$$

satisfies

$$f(\lambda)\leqq\phi(\lambda) \tag{2.11}$$

for all $0\leqq\lambda\leqq\lambda^*$. Let $\eta=-f'(0)/\gamma$ denote the minimizer of $\phi(\lambda)$; then η satisfies the inequalities $\eta\leqq\lambda^*$, $\eta\leqq\hat{\lambda}$, and $\eta/2\leqq\lambda^{k,j}$. (The last inequality is based on the assumption that $\omega\geqq1$. If $0<\omega<1$, then γ should be replaced by γ/ω.) Recall also that $\lambda^{k,j}\leqq\lambda^*$. Hence the inequalities

$$\phi\left(\frac{\eta}{2}\right)\geqq f\left(\frac{\eta}{2}\right)\geqq f(\lambda^{k,j}) \tag{2.12}$$

imply

$$f(0)-f(\lambda^{k,j})\geqq f(0)-f\left(\frac{\eta}{2}\right)\geqq f(0)-\phi\left(\frac{\eta}{2}\right)=\phi(0)-\phi\left(\frac{\eta}{2}\right) \tag{2.13}$$

or

$$(F(\mathbf{x}^{k,j}))^p-(F(\mathbf{x}^{k,j+1}))^p\geqq(\tfrac{3}{8})(f'(0))^2/\gamma. \tag{2.14}$$

Consequently, $|f'(0)|$ tends to zero as k increases.

Our next task is to show that $\lambda^{k,j}$ tends to zero as k increases. Assume for simplicity that $N_j=\{1,2,\cdots,\nu\}$ and let the vectors $\mathbf{u}=(u_1,\cdots,u_\nu)^T$ and $\mathbf{v}=(v_1,\cdots,v_\nu)^T$ be defined by the rules $u_i=w_i\alpha_i^2$ and $v_i=\beta_i/\alpha_i$. Then

$$\hat{\lambda}=-\mathbf{u}^T\mathbf{v}/((p-1)\|\mathbf{u}\|_1) \tag{2.15}$$

while the Hölder inequality implies

$$|\hat{\lambda}|\leqq\|\mathbf{v}\|_\infty. \tag{2.16}$$

From this point, the proof continues by contradiction. Assume that there exists a subsequence of $\{\lambda^{k,j}\}$ and a positive constant, say, $\delta>0$, for which the inequality $\delta<|\lambda^{k,j}|$ holds. Defining

$$\alpha_{\min}=\min\{|\alpha_i| \mid i\in N_j\} \tag{2.17}$$

and

$$\beta_{\max}=\max\{|\beta_i| \mid i\in N_j\}, \tag{2.18}$$

we then have that (2.16) implies

$$\delta\omega^{-1}\alpha_{\min}\leqq\beta_{\max} \tag{2.19}$$

and

$$f''(0) \geqq p(p-1) \sum_{i \in N_j} |\beta_i|^{p-2}\alpha_{\min}^2 \geqq \beta_{\max}^{p-2}\alpha_{\min}^2 \geqq (\delta\omega^{-1}\alpha_{\min})^{p-2}\alpha_{\min}^2 = \alpha_{\min}^p \delta^{p-2}\omega^{2-p}. \tag{2.20}$$

That is, $f''(0)$ exceeds a certain positive constant. Therefore, since $f'(0)$ tends to zero, $\hat{\lambda}$ also tends to zero, which contradicts the assumption that $\omega\hat{\lambda} \geqq \lambda^{k,j} > \delta$.

Now that we have proved that both $f'(0)$ and $\lambda^{k,j}$ tend to zero, the continuity of the first derivatives of $F(\mathbf{x})$ indicates that the gradient vector of $F(\mathbf{x})$ vanishes at any accumulation point of the sequence $\{\mathbf{x}^{k,j}\}$. Recall also that $F(\mathbf{x})$ is a convex twice-continuously differentiable function. Therefore, any point at which the gradient vector vanishes is a minimizer of $F(\mathbf{x})$. □

COROLLARY. *If $F(\mathbf{x})$ is strictly convex (i.e., the columns of A are linearly independent), it has a unique minimizer and the sequence $\{\mathbf{x}^{k,j}\}$ converges to this point.*

3. Column relaxation for $1<p<2$. In this case, the objective function to be minimized is the HAP function. The proof of (1.12) is based on the following inequalities:

$$\begin{aligned} F(\mathbf{x}) \leqq H(\mathbf{x}) &= \left\{ \sum_{i=1}^m [(|\mathbf{a}_i^T\mathbf{x} - \mathbf{b}_i|^2 + \varepsilon^2)^{1/2}]^p \right\}^{1/p} \\ &\leqq \left\{ \sum_{i=1}^m [|\mathbf{a}_i^T\mathbf{x} - b_i| + \varepsilon]^p \right\}^{1/p} \\ &\leqq \left(\sum_{i=1}^m |\mathbf{a}_i^T\mathbf{x} - b_i|^p \right)^{1/p} + \left(\sum_{i=1}^m \varepsilon^p \right)^{1/p} \\ &= F(\mathbf{x}) + m^{1/p}\varepsilon \end{aligned} \tag{3.1}$$

where the last inequality is the Minkowski inequality. It is also easy to verify that the HAP function is convex, and strictly convex if and only if the columns of A are linearly independent. The advantage of using $H(\mathbf{x})$ instead of $F(\mathbf{x})$ is that for fixed $\varepsilon > 0$, $H(\mathbf{x})$ has continuous second derivatives that are uniformly bounded in $\mathbb{R}^n$. The extension of the column relaxation method to minimize $H(\mathbf{x})$ is straightforward. Here the one parameter function $f(\lambda)$ is replaced by

$$h(\lambda) = \sum_{i \in N_j} [(\alpha_i\lambda + \beta_i)^2 + \varepsilon^2]^{p/2} \tag{3.2}$$

where N_j, α_i, and β_i are the same as before. Consequently,

$$h'(\lambda) = p \sum_{i \in N_j} \alpha_i(\alpha_i\lambda + \beta_i)[(\alpha_i\lambda + \beta_i)^2 + \varepsilon^2]^{-(2-p)/2} \tag{3.3}$$

and

$$h''(\lambda) = p \sum_{i \in N_j} \gamma_i(\lambda)\alpha_i^2[(\alpha_i\lambda + \beta_i)^2 + \varepsilon^2]^{-(2-p)/2} \tag{3.4}$$

where

$$\gamma_i(\lambda) = 1 - (2-p)(\alpha_i\lambda + \beta_i)^2/[(\alpha_i\lambda + \beta_i)^2 + \varepsilon^2]. \tag{3.5}$$

The Gauss-Seidel steplength is

$$\hat{\lambda} = -\frac{h'(0)}{h''(0)}, \tag{3.6}$$

and $\lambda^{k,j}$ is the first number in the sequence

$$\rho_\ell = (\tfrac{1}{2})^\ell \omega \hat{\lambda}, \qquad \ell = 0, 1, 2, \cdots \tag{3.7}$$

that satisfies

$$h'(\rho_\ell) \leqq 0. \tag{3.8}$$

THEOREM 2. *The above column relaxation method is well defined. The sequence $\{H(\mathbf{x}^{k,j})\}$ is monotone decreasing, $\lim_{k\to\infty} |\lambda^{k,j}| = 0$ for $j = 1, \cdots, n$, and any cluster point of the sequence $\{\mathbf{x}^{k,j}\}$ is a minimizer of $H(\mathbf{x})$. Furthermore, if the columns of A are linearly independent, $H(\mathbf{x})$ has a unique minimizer and the sequence $\{\mathbf{x}^{k,j}\}$ converges to this point.*

The proof of these results is similar to that of Theorem 1. In fact, it is somewhat simpler since the bounds

$$p - 1 \leqq \gamma_i(\lambda) \leqq 1 \tag{3.9}$$

and

$$0 \leqq h''(\lambda) \leqq p\nu\alpha\varepsilon^{p-2} \tag{3.10}$$

hold for all $\lambda \in \mathbb{R}$.

4. Column relaxation for ℓ_1 least norm problems. In this case

$$f(\lambda) = \sum_{i \in N_j} |\alpha_i \lambda + \beta_i| \tag{4.1}$$

and any minimizer of $f(\lambda)$ lies at one of the "breakpoints"

$$\lambda_i = -\frac{\beta_i}{\alpha_i}, \qquad i \in N_j. \tag{4.2}$$

Therefore we define

$$\lambda^{k,j} = \lambda_\ell \tag{4.3}$$

where

$$f(\lambda_\ell) = \min \{ f(\lambda_i) \mid i \in N_j \}. \tag{4.4}$$

(If the minimizer is not unique, we choose the smallest one.)

THEOREM 3. *The sequence $\{\mathbf{x}^{k,j}\}$ that is generated by the above column relaxation method always converges. However, it may converge to a nonoptimal point.*

Proof. Assume for simplicity that $N_j = \{1, 2, \cdots, \nu\}$ and $\lambda_1 \leqq \lambda_2 \leqq \cdots \leqq \lambda_\nu$. Then

$$f'(\lambda) = \sum_{i \in N_j} \alpha_i \operatorname{sign}(\alpha_i \lambda + \beta_i) \tag{4.5}$$

has a constant value at each of the open intervals $(\lambda_i, \lambda_{i+1})$, $i = 0, 1, \cdots, \nu$, where $\lambda_0 = -\infty$ and $\lambda_{\nu+1} = \infty$. It therefore follows that $f'(\lambda)$ can obtain a finite number of values, regardless of the choice of β_i, $i \in N_j$. Hence, there exists a positive constant, say δ, such that $f'(\lambda) \neq 0$ implies $|f'(\lambda)| > \delta$. It is also possible to assume that $\lambda^{k,j} > 0$. In this case, $f(\lambda)$ is strictly decreasing in the interval $[0, \lambda^{k,j}]$ and

$$f'(\lambda) \leqq -\delta \tag{4.6}$$

for any point $\lambda \neq \lambda_i$ in this interval. The last inequality yields

$$f(0) - f(\lambda^{k,j}) \geqq \delta |\lambda^{k,j}|. \tag{4.7}$$

Hence the relations

$$F(\mathbf{x}^{k,j}) - F(\mathbf{x}^{k,j+1}) = f(0) - f(\lambda^{k,j}) \tag{4.8}$$

and

$$F(\mathbf{x}^{1,1}) - F(\mathbf{x}^{q+1,1}) = \sum_{k=1}^{q} \sum_{j=1}^{n} F(\mathbf{x}^{k,j}) - F(\mathbf{x}^{k,j+1}) \tag{4.9}$$

imply the convergence of the series

$$\sum_{k=1}^{\infty} \sum_{j=1}^{n} |\lambda^{k,j}|, \tag{4.10}$$

which means that $\{\mathbf{x}^{k,j}\}$ also converges.

The following example illustrates that the sequence $\{\mathbf{x}^{k,j}\}$ may stick at a nonoptimal point: Consider the system $2x+2y=0$ and $x-y=0$, whose unique solution is the origin point. Yet the above algorithm is not able to move out of the point $(-1, 1)$, or any other point of the form $(-\Theta, \Theta)$ where $\Theta \neq 0$. □

A simple remedy that enables us to overcome the above difficulty is to minimize the HAP function instead of $F(\mathbf{x})$. Note that here

$$H(\mathbf{x}) = \sum_{i=1}^{m} [(\mathbf{a}_i^T\mathbf{x} - b_i)^2 + \varepsilon^2]^{1/2}, \tag{4.11}$$

$$h(\lambda) = \sum_{i \in N_j} [(\alpha_i\lambda + \beta_u)^2 + \varepsilon^2]^{1/2}, \tag{4.12}$$

$$h'(\lambda) = \sum_{i \in N_j} \alpha_i(\alpha_i\lambda + \beta_i)/[(\alpha_i\lambda + \beta_i)^2 + \varepsilon^2]^{1/2}, \tag{4.13}$$

$$h''(\lambda) = \varepsilon^2 \sum_{i \in N_j} \alpha_i^2/[(\alpha_i\lambda + \beta_i)^2 + \varepsilon^2]^{3/2}. \tag{4.14}$$

The structure of $h''(\lambda)$ indicates that it is bounded above by $(\sum_{i\in N_j} \alpha_i^2)/\varepsilon$. Also, by repeating the arguments mentioned in the proof of Theorem 1, it is easy to show that $h''(\lambda)$ is bounded from below by a certain multiple of $H(\mathbf{x}^{1,1})$, in the interval $[0, \lambda^*]$. Consequently, Theorem 2 remains true in this case. Of course, once a minimizer of the HAP function is identified, it can be refined by the method of this section. However, as Theorem 3 indicates, there is no guarantee that this will result in an ℓ_1 solution.

5. Column relaxation for ℓ_∞ least norm problems. In this case

$$f(\lambda) = \max_{i \in N_j} |\alpha_i\lambda + \beta_i|. \tag{5.1}$$

Thus, as for $p=1$, it is a convex piecewise linear function whose minimizer lies at one of its breakpoints. Hence, $\lambda^{k,j}$ is defined to be a minimizer of $f(\lambda)$. The search for the minimizer starts by checking whether the origin itself is a minimizer. If not, we move to the closest breakpoint that decreases $f(\lambda)$. If this point is not optimal, we move to the next closest breakpoint that decreases $f(\lambda)$. In this way, the search proceeds from one breakpoint to the next until a minimizer is found. The details of the method are left to the reader. (A similar line-search algorithm is described in Bartels, Conn, and Charalambous [1] in the context of dense ℓ_∞ problems.)

THEOREM 4. *The sequence $\{\mathbf{x}^{k,j}\}$ that is generated by the above column relaxation method always converges. However, it may converge to a nonoptimal point.*

The proof of Theorem 4 is similar to that of Theorem3. Here inequality (4.6) is replaced by

$$|f'(\lambda)| \geqq \min_{i \in N_j} |\alpha_i|, \tag{5.2}$$

which proves the first claim. The second claim is proved by considering the system $2x-y=2$ and $-x+2y=2$, whose unique solution is $x=y=2$. In this example, the algorithm is not able to move out of the origin point.

Here again there is a simple way to overcome the above difficulty. Let $\mathbf{x}^{(p)}$ and $\mathbf{x}^{(\infty)}$ denote solutions of (1.1) when $1 \leqq p < \infty$ and $p=\infty$, respectively. Then it can be shown that

$$\|A\mathbf{x}^{(p)}-\mathbf{b}\|_p \geqq \|A\mathbf{x}^{(p)}-\mathbf{b}\|_\infty \geqq \|A\mathbf{x}^{(\infty)}-\mathbf{b}\|_\infty \tag{5.3}$$

and

$$\lim_{p\to\infty} \|A\mathbf{x}^{(p)}-\mathbf{b}\|_p = \|A\mathbf{x}^{(\infty)}-\mathbf{b}\|_\infty. \tag{5.4}$$

Moreover, if the columns of A are linearly independent, then

$$\lim_{p\to\infty} \mathbf{x}^{(p)} = \mathbf{x}^{(\infty)}. \tag{5.5}$$

Thus, for large values of p, $\mathbf{x}^{(p)}$ is a good substitute for $\mathbf{x}^{(\infty)}$ (see Cheney [5, p. 43]). In practice, it is sufficient to estimate $\mathbf{x}^{(p)}$ for a moderate value of p (e.g., $p=10$), using the method of § 2. Then, at the second stage, the resulting estimate can be refined by the method of this section. However, as Theorem 4 indicates, there is no guarantee that this will result in an ℓ_∞ solution.

6. Further remarks. We will start with a few words on the choice of an initial point. The column relaxation methods that are described in §§ 2 and 3 are based on a second-order Taylor expansion of the objective function. The relationship of these methods to the SOR method indicates that near a minimizer the rate of convergence is expected to be linear. On the other hand, far from a minimizer, the Taylor expansion is of limited value and the methods are expected to show slow progress. An exception to this rule occurs when $p=2$. In this case, the column relaxation iteration (1.8) coincides with the SOR iteration for solving (1.7) and, consequently, has a linear rate of convergence. Recall also that the basic properties of vector norms imply that $\mathbf{x}^{(2)}$ is likely to provide a reasonable starting point. Therefore, if a better initial guess is not available, it is recommended to start the minimization process with a few iterations of the column relaxation method for solving the least squares problem

$$\text{minimize } \|A\mathbf{x}-\mathbf{b}\|_2^2. \tag{6.1}$$

If the starting point $\mathbf{x}^{(1,1)}$ is far from $\mathbf{x}^{(2)}$, the SOR method may require many iterations to reach $\mathbf{x}^{(2)}$. (The same is true whenever $\|\mathbf{x}^{(1,1)}\|$ is much larger than $\|\mathbf{x}^{(2)}\|$.) Fortunately, there is a simple scaling process that enables us to overcome this deficiency. The scaling operation takes place at the end of an SOR iteration. Let $\mathbf{x}$ denote the current point and let $\mathbf{r}=A\mathbf{x}-\mathbf{b}$ denote the corresponding residual vector. Then the one-parameter function

$$\psi(\Theta) = \|A(\Theta\mathbf{x})-\mathbf{b}\|_2^2 \tag{6.2}$$

has a unique minimizer at the point

$$\hat{\Theta} = (\mathbf{r}+\mathbf{b})^T\mathbf{b}/(\mathbf{r}+\mathbf{b})^T(\mathbf{r}+\mathbf{b}). \tag{6.3}$$

Thus, changing $\mathbf{x}$ to $\hat{\Theta}\mathbf{x}$ is likely to reduce the objective function value. Note that the corresponding change in $\mathbf{r}$ is $(\hat{\Theta}-1)(\mathbf{r}+\mathbf{b})$.

As p increases, the nonlinearity of $F(\mathbf{x})$ increases and the convergence of the column relaxation method is expected to slow down. Hence if p is much larger than two, it may be advantageous to approach the solution gradually. That is, the problem (1.1) is solved a few times, each time with an increasing value of p, and the solution of one problem serves as a starting point for the next problem.

A similar remark applies to the HAP function. In using this function to solve multifacility location problems, it has been observed that the larger the value of ε the faster the convergence. Consequently, it is recommended to start with a relatively large value of ε, and to reduce this value successively (see Eyster, White, and Wierwille [12]). In our case, the initial value of ε need not be larger than $\|\mathbf{r}\|_1/m$, whereas the final value need not be smaller than $\|\mathbf{r}\|_1/(100m)$, where $\mathbf{r}$ denotes the current residual vector. Observe also that if $p=1$ and ε is much smaller than $\|\mathbf{r}\|_1/m$, then the initial steplength of the line search can be much larger than necessary.

The theory of the nonlinear SOR method suggests that a well-tuned scheme should run faster than the Gauss-Seidel scheme. However, in practice, the computation of an optimal relaxation parameter is not an easy task, since neither the solution point nor the corresponding second derivatives matrix are known in advance. (Recall also that A is assumed to be a sparse, unstructured matrix.) Thus, in many problems, the only practical way to improve the relaxation parameter ω is by performing repeated runs with various values of ω. On the other hand, the relation between the column relaxation method and the method that changes one variable at a time suggests another acceleration technique. The experience that has been gained in solving small problems indicates that lines that join successive iterants are likely to provide good descent directions (see Fletcher, [13, p. 15]). Hence the basic algorithm can be modified by performing an extra line search every few iterations. In this way, the iterations of the modified algorithm are divided into "cycles," where each cycle is composed of a fixed number, say c, of iterations. Let the n-vectors $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ denote the current solutions at the beginning and the end of such a cycle. Then the starting point for the next cycle is defined as $\mathbf{x}^{(2)}+\Theta(\mathbf{x}^{(2)}-\mathbf{x}^{(1)})$, where Θ estimates a minimizer of the one-parameter function

$$\psi(\Theta)=F(\mathbf{x}^{(2)}+\Theta(\mathbf{x}^{(2)}-\mathbf{x}^{(1)})). \tag{6.4}$$

The usefulness of this technique in solving large minimization problems is illustrated by Dax in [7] and [8]. It is noted there that the modified algorithm can be viewed as a truncated Newton method that uses a fixed number of SOR iterations to solve the Newton equation. In our case, $\psi(\Theta)$ has the form

$$\begin{aligned}\psi(\Theta)&=\|A(\mathbf{x}^{(2)}+\Theta(\mathbf{x}^{(2)}-\mathbf{x}^{(1)}))-\mathbf{b}\|_p\\&=\|\Theta A(\mathbf{x}^{(2)}-\mathbf{x}^{(1)})+\mathbf{r}^{(2)}\|_p\\&=\|\Theta(\mathbf{r}^{(2)}-\mathbf{r}^{(1)})+\mathbf{r}^{(2)}\|_p\end{aligned} \tag{6.5}$$

where $\mathbf{r}^{(1)}$ and $\mathbf{r}^{(2)}$ denote the corresponding residual vectors. Hence the calculation of Θ needs no matrix-vector product. However, since the vector $\mathbf{r}^{(2)}-\mathbf{r}^{(1)}$ may have many nonzero components, this process requires special consideration.

There are many large scale ℓ_p problems in which the objective function is associated with some grid or mesh in R^k, $k=1$, 2, or 3 (e.g., Examples 2 and 3 of § 7). This suggests the use of acceleration techniques that are based on a variable mesh size. One such technique is the well-known multigrid method (e.g., Brandt [3]). A less sophisticated technique is the Successive Mesh Refinement Method, in which we gradually increase the size of the mesh, using the solution on one mesh as a starting point for the next (finer) mesh. (The usefulness of this idea is illustrated by Dax [7], who used it to accelerate the nonlinear Gauss-Seidel method.) It is also worthwhile to note that the proposed methods are easily modified to allow simple bounds on the variables. This can be done by following the ideas of Cryer [6].

The next section describes numerical experiments with the methods proposed in the previous sections. However, as this is an initial work on relaxation methods for solving ℓ_p problems with $p \neq 2$, these experiments were carried out without testing the above ideas.

7. Numerical results. To illustrate the practical usefulness of the proposed methods, we have designed two algorithms; one solves the ℓ_1 least norm problem, and the other solves the ℓ_∞ least norm problem. The first algorithm is composed of three stages. The first stage performs 10 iterations of the Gauss-Seidel method for solving (6.1). The second stage performs 10 iterations of the column relaxation method for minimizing the HAP function (4.11). The third stage performs 10 iterations of the method described in the beginning of § 4.

The second algorithm, which is designed to solve the ℓ_∞ least norm problem, is also composed of three stages. The first is the same as above. The second stage performs 10 iterations of the method described in § 2, using $p = 10$. The third stage performs 10 iterations of the method described in the beginning of § 5.

Note that both algorithms are heuristic in the sense that the third stage may yield a nonoptimal point. It was found sufficient to limit the number of iterations performed at each stage to 10, since increasing the number of iterations did not result in a substantial improvement in the final value of the objective function. Also, for the same reason, there was no need to use a larger value for p. The initial value of ε was $\|\mathbf{r}\|_1/m$, where $\mathbf{r}$ denotes the residual vector at the end of the first stage. The value of ε was halved every two iterations.

Both algorithms were programmed in FORTRAN, using single precision arithmetic. The first two examples were run on a CDC CYBER 170-855 computer, whereas the third example was run on an IBM personal computer.

The results of our experiments are given in Tables 1-6. The figures in these tables provide the values of $\|A\mathbf{x}-\mathbf{b}\|_1$, $\|A\mathbf{x}-\mathbf{b}\|_2$, $\|A\mathbf{x}-\mathbf{b}\|_{10}$, and $\|A\mathbf{x}-\mathbf{b}\|_\infty$ at the end of each stage. (The starting point is referred to as Stage 0.) The type of the objective function that is minimized at each stage appears under the heading "Type."

Example 1. *Random test problems.* In this example, $m = 2{,}000$ and $n = 1{,}000$. The components of $\mathbf{b}$ are random numbers from the interval $[-1, 1]$. The locations and the values of the nonzero elements of A are defined in the following way. As before, we use $\mathbf{a}_i^T$ to denote the ith row of A, and μ_i to denote the number of nonzero elements in $\mathbf{a}_i$, $i = 1, \cdots, m$. Then μ_i is a random number from the set $\{1, 2, 3, 4, 5\}$. Similarly, the locations of the nonzero elements of $\mathbf{a}_i$ are (distinct) random numbers from the set $\{1, 2, \cdots, n\}$, whereas the values of these elements are random numbers from the interval $[-1{,}000,\ 1{,}000]$. (All the random number generators are of uniform distribution.) The starting point is $(1, 1, \cdots, 1)^T \in \mathbb{R}^n$, which is a poor initial guess. Hence the convergence of the Gauss-Seidel process was accelerated by applying the scaling (6.3). The results are given in Tables 1-2.

Example 2. *The calculation of a linear spline.* This example considers the following data-fitting (or data-compressing) problem. Suppose that we have m data pairs $\{(p_i, b_i)\}$, $i = 1, \cdots, m$, of real numbers and we assume they have been sorted so that $p_1 \leqq p_2 \leqq \cdots \leqq p_m$. The data points $\{p_i\}$ lie in a given interval $[\alpha, \beta]$, whereas the data observations $\{b_i\}$ denote the corresponding values of a real function (e.g., a physical phenomenon) at these points. It is desired to fit the data by a continuous function $s(p)$, whose representation reduces the storage needed to represent the data.

Probably the simplest way to achieve this task is to construct a continuous piecewise linear function with given breakpoints

$$\alpha = \eta_1 < \eta_2 < \cdots < \eta_n = \beta.$$

Let the vector of unknowns $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T \in \mathbb{R}^n$ denote the corresponding values

TABLE 1
The ℓ_1 solution of a random test problem.

Stage	Type	ℓ_1	ℓ_2	ℓ_∞
0		1,571,667.79	44,792.14	4,271.666
1	ℓ_2	642.24	18.07	1.296
2	HAP	562.17	20.97	2.288
3	ℓ_1	558.88	20.23	2.293

TABLE 2
The ℓ_∞ solution of a random test problem.

Stage	Type	ℓ_1	ℓ_2	ℓ_{10}	ℓ_∞
0		1,571,667.79	44,792.14		4,271.666
1	ℓ_2	642.24	18.07	1.576	1.296
2	ℓ_{10}	816.12	20.40	1.310	1.008
3	ℓ_∞	977.89	24.50		0.898

TABLE 3
The ℓ_1 calculation of a line spline.

Stage	Type	ℓ_1	ℓ_2	ℓ_∞
0		13,706.313	397.451	23.4639
1	ℓ_2	9.181	0.316	0.0429
2	HAP	8.735	0.335	0.0518
3	ℓ_1	8.718	0.337	0.0515

TABLE 4
The ℓ_∞ calculation of a line spline.

Stage	Type	ℓ_1	ℓ_2	ℓ_{10}	ℓ_∞
0		13,706.313	397.451		23.4639
1	ℓ_2	9.181	0.316	0.0453	0.0429
2	ℓ_{10}	10.473	0.337	0.0370	0.0298
3	ℓ_∞	11.942	0.367		0.0297

TABLE 5
The ℓ_1 calculation of a spline surface.

Stage	Type	ℓ_1	ℓ_2	ℓ_∞
0		164.917	24.340	8.782
1	ℓ_2	17.690	2.528	0.858
2	HAP	17.422	2.503	0.933
3	ℓ_1	15.818	2.790	1.035

TABLE 6
The ℓ_∞ calculation of a spline surface.

Stage	Type	ℓ_1	ℓ_2	ℓ_{10}	ℓ_∞
0		164.917	24.340		8.782
1	ℓ_2	17.690	2.528	0.874	0.858
2	ℓ_{10}	22.636	2.819	0.692	0.594
3	ℓ_∞	23.570	2.903		0.589

of $s(p)$ at these points. Then for a given point $p \in [\alpha, \beta]$, the value of $s(p)$ is defined by

$$s(p) = (1-\Theta)x_j + \Theta x_{j+1}$$

where j is an index such that $\eta_j \leqq p \leqq \eta_{j+1}$ and $\Theta = (p-\eta_j)/(\eta_{j+1}-\eta_j)$. It is possible to obtain values $\{x_j\}$ by solving a least norm problem

$$\min \sum_{i=1}^{m} |s(p_i) - b_i|^p.$$

In the current example $m = 2{,}000$ and the data points are random numbers from the interval $[0, 10]$. The observation points are defined by the rule $b_i = \exp(\sqrt{p_i}) \sin(3p_i)$. The data interval was evenly partitioned into 200 subintervals, so that $n = 201$, and $\eta_j = (j-1)0.05, j = 1, \cdots, n$. The starting point is $\mathbf{0} = (0, 0, \cdots, 0)^T \in \mathbb{R}^n$. The results are presented in Tables 3-4.

Example 3. *Fitting a spline surface to scattered data.* This example considers the calculation of a two-dimensional line spline. Here, we are given m triplets of data (x_i, y_i, z_i), $i = 1, \cdots, m$, where the datapoints, (x_i, y_i), are scattered arbitrarily inside a rectangle, say R, in the (x, y) plane. The data observations, z_i, denote the height of an unknown "surface," say $f(x, y)$, at the datapoints. The rectangle R is divided into a rectangular mesh of points, and we wish to estimate the surface height at these points. This problem arises in many applications. For example, there are algorithms for plotting contour maps and three-dimensional "fishnet" surface maps that consist of two stages. The first stage calculates the surface height on a rectangular mesh, whereas the second stage draws the contours or the "fishnet."

Assume that $R = [a, b] \times [c, d]$ and let the intervals $[a, b]$ and $[c, d]$ be partitioned by the points

$$a = \eta_1 < \eta_2 < \cdots < \eta_s = b$$

and

$$c = \sigma_1 < \sigma_2 < \cdots < \sigma_t = d,$$

respectively. Then the points (η_i, σ_j) form a rectangular grid that partitions R into $(s-1) \times (t-1)$ subrectangles. We will use R_{ij} to denote the (i, j) subrectangle. That is,

$$R_{ij} = \{(x, y) \mid \eta_i \leqq x \leqq \eta_{i+1} \text{ and } \sigma_j \leqq y \leqq \sigma_{j+1}\}.$$

If $f(x, y)$ is evaluated by a two-dimensional line spline, say $S(x, y)$, then the value of S at a point $(x, y) \in R_{ij}$ is given by the formula

$$S(x, y) = S_{ij}v_1 + S_{i+1,j}v_2 + S_{i,j+1}v_3 + S_{i+1,j+1}v_4$$

where $S_{ij} = S(\eta_i, \sigma_j)$ denotes the value of $S(x, y)$ at the (i, j) gridpoint, and $v_\ell = v_\ell(x, y)$, $\ell = 1, 2, 3, 4$, denote the values of the corresponding "basis functions" at the point (x, y). Defining

$$\phi_i = \phi_i(x) = (x - \eta_i)/(\eta_{i+1} - \eta_i),$$

$$\psi_j = \psi_j(x) = (y - \sigma_j)/(\sigma_{j+1} - \sigma_j),$$

then

$$v_1 = (1-\phi_i)(1-\psi_j), \qquad v_2 = \phi_i(1-\psi_j),$$

$$v_3 = \phi_i\psi_j, \qquad v_4 = (1-\phi_i)\psi_j$$

(see any textbook on finite-element methods). The residual vector $\mathbf{r} = (r_1, \cdots, r_m)^T$ whose norm we wish to minimize is defined by the relation

$$r_k = S(x_k, y_k) - z_k, \qquad k = 1, \cdots, m.$$

Therefore, since $S(x_k, y_k)$ is a linear combination of the unknown values $\{S_{ij}\}$, the problem has the form (1.1) where A has m rows and $n = s \cdot t$ columns. The kth row of A refers to the kth datapoint and contains only four nonzero elements, which are essentially the corresponding coefficients v_ℓ, $\ell = 1, 2, 3, 4$. In minimizing the objective function as a function of S_{ij}, it is necessary to know only the datapoints that lie in the "neighboring" subrectangles. Hence, here it is advantageous to order the data such that all the points that lie in the same subrectangle are grouped together, using a simple key that enables us to find the location of these points.

In the current example $m = 2{,}000$, and the datapoints are randomly spaced in the square region

$$R = \{(x, y) \mid -5 \leqq x \leqq 5, -5 \leqq y \leqq 5\}.$$

The surface height at the point (x_k, y_k) is

$$z_k = 10 \sin(x_k) \sin(y_k).$$

The region R was evenly spaced into 20 rows and 10 columns of subsquares. That is,

$$\eta_i = -1 + (i-1)0.5, \qquad i = 1, \cdots, 21,$$

$$\sigma_j = -1 + (j-1)0.5, \qquad j = 1, \cdots, 21.$$

Consequently, the problem has 441 unknowns, S_{ij}, which are the values of the spline surface at the points (η_i, σ_j). The initial value of S_{ij} is the average of all the data values, z_k, such that (x_k, y_k) lies in a "neighboring" subsquare. The results are given in Tables 5-6.

8. Concluding remarks.

The results of our experiments illustrate the ability of the proposed methods to handle large ℓ_1 and ℓ_∞ problems. Starting from an arbitrary point, the algorithm calculated a good estimate of the solution within a few iterations. The new method is simple and robust, it requires a minimal amount of computer storage, and the computational effort per iteration is small.

REFERENCES

[1] R. H. BARTELS, A. R. CONN, AND C. CHARALAMBOUS, *On Cline's direct method for solving overdetermined linear systems in the ℓ_∞ sense*, SIAM J. Numer. Anal., 15 (1978), pp. 255-270.

[2] A. BJÖRCK AND T. ELFVING, *Accelerated projection methods for computing pseudoinverse solutions of systems of linear equations*, BIT, 19 (1979), pp. 145-163.

[3] A. BRANDT, *Algebraic multigrid theory: The symmetric case*, in Proc. Internat. Multigrid Conference, Copper Mountain, CO, April 1983.

[4] Y. CENSOR, *Row-action methods for huge and sparse systems and their applications*, SIAM Rev., 23 (1981), pp. 444-466.

[5] E. W. CHENEY, *Introduction to Approximation Theory*, McGraw-Hill, New York, 1966.

[6] C. W. CRYER, *The solution of a quadratic programming problem using systematic overrelaxation*, SIAM J. Control, 9 (1971), pp. 385-392.

[7] A. DAX, *Successive refinement of large multicell models*, SIAM J. Numer. Anal., 22 (1985), pp. 865-887.

[8] ———, *Line search acceleration of iterative methods*, Tech. Report, Hydrological Service of Israel, Jerusalem, Israel, 1988; Linear Algebra Appl., to appear.

[9] ———, *The convergence of linear stationary iterative processes for solving singular unstructured systems of linear equations*, Tech. Report, Hydrological Service of Israel, Jerusalem, Israel, 1989; SIAM Rev., to appear.

[10] ———, *Row relaxation methods for least norm problems*, Tech. Report, Hydrological Service of Israel, Jerusalem, Israel, 1990, in preparation.

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

[12] J. W. EYSTER, J. A. WHITE, AND W. W. WIERWILLE, *On solving multifacility location problems using a hyperboloid approximation procedure*, AIIE Trans., 5 (1973), pp. 1-6.

[13] R. FLETCHER, *Practical Methods of Optimization, Vol. 1: Unconstrained Optimization*, John Wiley, Chichester, 1980.

[14] R. L. FRANCIS AND J. A. WHITE, *Facility Layout and Location, an Analytical Approach*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[15] J. M. ORTEGA AND W. C. RHEINBOLDT, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970.

[16] M. J. D. POWELL, *On search directions for minimization algorithms*, Math. Programming, 4 (1973), pp. 193-201.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 5, pp. 990–1028, September 1990

QR FACTORIZATION OF A DENSE MATRIX ON A HYPERCUBE MULTIPROCESSOR*

ELEANOR CHU[†] AND ALAN GEORGE[‡]

Abstract. In this article a new algorithm for computing the QR factorization of a rectangular matrix on a hypercube multiprocessor is described. The hypercube network is configured as a two-dimensional subcube-grid in the proposed scheme. A global communication scheme that uses redundant computation to maintain data proximity is employed, and the mapping strategy is such that for a fixed number of processors the processor idle time is small and either constant or grows linearly with the dimension of the matrix. A complexity analysis shows what the aspect ratio of the configured grid should be in terms of the shape of the matrix and the relative speeds of communication and computation. Numerical experiments performed on an Intel Hypercube multiprocessor support the theoretical results.

Key words. QR factorization, parallel computation, hypercube multiprocessors

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

1. Introduction.

In this article we present an algorithm for reducing an $m \times n$ matrix to upper triangular form using orthogonal transformations on a hypercube multiprocessor. The hypercube network is configured as a two-dimensional *subcube-grid* in the proposed scheme. 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 are required to form a hypercube of smaller dimension, which is a subcube. The apparent difference between the two is illustrated by an example in Fig. 1, where each processor is denoted by a "•" and each physical communication channel is represented by a solid line between two processors. The diagram on the left in Fig. 1 shows a 4×4 subcube-grid and the diagram on the right shows a 4×4 mesh-connected processor array. More details on the subcube-grid configuration and its advantage are presented after we introduce the hypercube network in §2.

For easy exposition, we first describe a special case of the algorithm in order to explain some basic strategies for data mapping and interprocessor communication on the hypercube. We refer to the special case as Algorithm I, and the general algorithm as Algorithm II. Finally we propose further enhancement to reduce both arithmetic and communication costs of Algorithm II. This version of the algorithm will be referred to as the enhanced Algorithm II.

Algorithms I and II are based on Givens rotations. The use of Givens transformations has been widely studied in the literature for parallel implementation on systolic

* Received by the editors January 14, 1988; accepted for publication (in revised form) July 7, 1989.

† Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1. The research of this author was supported in part by the Natural Sciences and Engineering Research Council of Canada under grant A8111.

‡ Department of Computer Science, University of Tennessee, Knoxville, Tennessee 37919. Present address, Needles Hall, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1. This author is also affiliated with the Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, Tennessee, through the University of Tennessee/Oak Ridge National Laboratory Distinguished Scientist program. The research of this author was supported in part by the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy under contract DE-AC05-84OR21400 with Martin Marietta Energy Systems Inc., and by The Science Alliance, a state-supported program at the University of Tennessee.

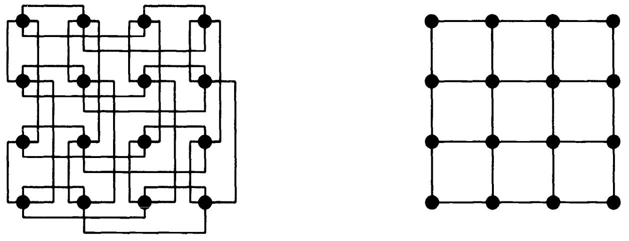


FIG. 1. *A* 4×4 *subcube-grid versus a* 4×4 *mesh-connected processor array.*

arrays [1], [2], [11], [14], [18], SIMD computers [19], shared-memory multiprocessors [5], [6], [8], [13], [15], and hypercube multiprocessors [3], [9], [16], [17]. When $m \geq n$, the mathematical computation we consider is usually formulated as

$$QA = \begin{pmatrix} R \\ 0 \end{pmatrix} ,$$

where A is an $m \times n$ matrix with full column rank, Q is an $m \times m$ orthogonal matrix, and R is an upper triangular matrix of order n. When $m < n$, we have

$$QA = (R\,S) ,$$

where A is an $m \times n$ matrix with full row rank, Q is as defined above, R is an upper triangular matrix of order m, and S is an $m \times (n - m)$ rectangular matrix. The matrix Q is formed as the product of Givens rotations such that the elements of A below the main diagonal are annihilated one at a time. Since there is much freedom in the order of applying the Givens rotations, the elements of A can be eliminated by many different orderings. The *independent* (or *disjoint*) rotations induced by a particular ordering can be computed simultaneously provided there are enough processors available. While the number of independent rotations increases with the problem size and changes during the factorization process, the number of processors on a parallel computer is fixed. Therefore, the independent rotations must be statically or dynamically allocated to the processors in some way. The choice of a different ordering and the particular strategy of assigning independent rotations to processors give rise to different parallel algorithms.

There are five parallel Givens algorithms proposed in [3], [9], [16], and [17] for hypercube implementation. They are all based on "Givens sequences," which are sequences of Givens rotations where zeros once created are preserved. The two new parallel algorithms proposed in this paper can be viewed as different implementations of a particular Givens sequence on the hypercube. Both algorithms take full advantage of the hypercube topology and require only *nearest-neighbor* communication. They differ from the algorithms in [3], [9], [16], and [17] in communication schemes, data mapping schemes, arithmetic/communication complexities, and work load distribution.

An outline of this paper is as follows. In §2, we introduce the communication network and the subcube-grid configuration of a hypercube multiprocessor. In §3, Algorithm I is described and examined in detail. Section 4 contains the performance analysis of Algorithm I. Algorithm II and its performance analysis are presented next in §§5 and 6. In §7, we report extensive numerical experiments and further enhancements to Algorithm II. Section 8 contains our concluding remarks.

2. Hypercube multiprocessors. There are 2^d identical node processors in a hypercube multiprocessor of dimension d. Each processor is uniquely identified by an integer in $\{0, 1, 2, \cdots, 2^d - 1\}$. If we represent each processor *id* in the set $\{0, 1, 2, \cdots, 2^d - 1\}$ by a d-bit binary string, $b_{d-1}b_{d-2}\cdots b_0$, then the hypercube network is constructed by physically connecting each pair of processors whose *id*'s differ in one single bit b_i, $0 \leq i \leq (d-1)$. Figure 2 illustrates the binary hypercube topologies of dimension $d = 1, 2$, and 3.

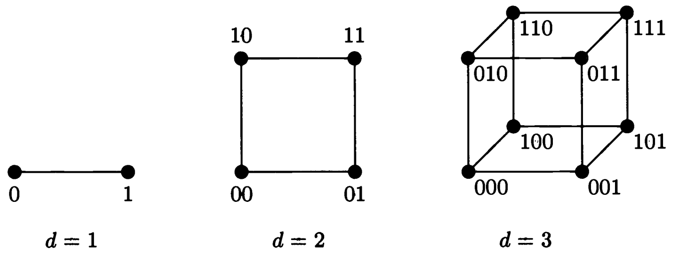


FIG. 2. *Binary hypercubes of dimension* 1, 2, *and* 3.

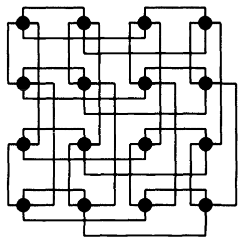
When the node processors on a local-memory multiprocessor cooperate with each other to solve a problem, they need to communicate data or synchronization information with each other by exchanging messages. Since the cost of communication increases with the number of messages, the length of each message, and the length of the path each message traverses, it is desirable to aim at reducing all of them in devising parallel algorithms.

2.1. The subcube-grid configurations. Algorithm II makes use of a $\gamma_1 \times \gamma_2$ subcube-grid configuration to reduce both arithmetic and communication cost. The objective of this configuration is to map the 2^d processors of a hypercube to a rectangular grid so that the processors in each column or each row of the grid form a subcube. Algorithm I is the special case of Algorithm II when $\gamma_1 = 2^d$ and $\gamma_2 = 1$, which is dealt with separately to help present the basic ideas more clearly. We emphasize that the description, analysis, and implementation of Algorithm II are general, and they apply to all possible choices of γ_1 and γ_2 for the subcube-grid, including the choice of Algorithm I.

If we let $d = d_1 + d_2$, $\gamma_1 = 2^{d_1}$, and $\gamma_2 = 2^{d_2}$, then a $\gamma_1 \times \gamma_2$ subcube-grid may be configured by requiring that the *id*'s of the processors in the same row differ in only the rightmost d_2 bits, and that the *id*'s of the processors in the same column differ in only the leftmost d_1 bits. This configuration corresponds to mapping processors to the $\gamma_1 \times \gamma_2$ grid row by row or column by column following the natural order of their processor *id*. Figure 3 displays a 4×4 subcube-grid consisting of 16 processors, and Fig. 4 displays an 8×4 subcube-grid consisting of 32 processors.

2.2. The communication algorithm. The communication algorithm we shall propose may be viewed as a *variant* of the subcube-doubling technique, which allows each processor to exchange a (probably different) message with all of its d neighbors after d communication steps. Using a hypercube of dimension $d = 3$, we illustrate in Fig. 5 the d communication steps in the basic subcube-doubling algorithm. To accomplish the d exchanges, each processor pairs with another processor whose *id* is different in bit $b_{\ell-1}$, $\ell = d, d-1, \cdots, 1$. For example, processor P_0 accomplishes the 3 exchanges by communicating with processors P_4, P_2, and P_1 sequentially. The processor *id*'s of the latter three processors are 100, 010, and 001, respectively. During

0000	0001	0010	0011
0100	0101	0110	0111
1000	1001	1010	1011
1100	1101	1110	1111



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}

FIG. 3. *The configuration of a* 4×4 *subcube-grid.*

00000	00001	00010	00011	P_0	P_1	P_2	P_3
00100	00101	00110	00111	P_4	P_5	P_6	P_7
01000	01001	01010	01011	P_8	P_9	P_{10}	P_{11}
01100	01101	01110	01111	P_{12}	P_{13}	P_{14}	P_{15}
10000	10001	10010	10011	P_{16}	P_{17}	P_{18}	P_{19}
10100	10101	10110	10111	P_{20}	P_{21}	P_{22}	P_{23}
11000	11001	11010	11011	P_{24}	P_{25}	P_{26}	P_{27}
11100	11101	11110	11111	P_{28}	P_{29}	P_{30}	P_{31}

FIG. 4. *The configuration of an* 8×4 *subcube-grid.*

each communication step, the 2^{d-1} pairs of processors exchange data concurrently using 2^{d-1} disjoint channels.

The basic subcube-doubling communication algorithm may be described for a d-dimension hypercube as below.

```
$\ell \leftarrow d$
**while** $\ell > 0$ **do**
    send (my message) to processor with *id* different
        from my id in bit $b_{\ell-1}$.
    receive a message
    update (my message) as instructed by each different algorithm
    $\ell \leftarrow \ell - 1$
```

Since each column and each row of processors in a $\gamma_1 \times \gamma_2$ subcube-grid are hypercubes of dimensions d_1 and d_2, the communication scheme as described above may be employed within each column and each row of processors if we initialize "$\ell \leftarrow d_1$" and "$\ell \leftarrow d_2$," respectively.

Different communication algorithms may also 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. For example, in some cases, "my message" may be simply overwritten by the message received after each exchange; in other cases, "my message" may be updated by the result of more sophisticated computation combining itself with the data received.

In this article we show how *redundant update* performed by otherwise idle pro-

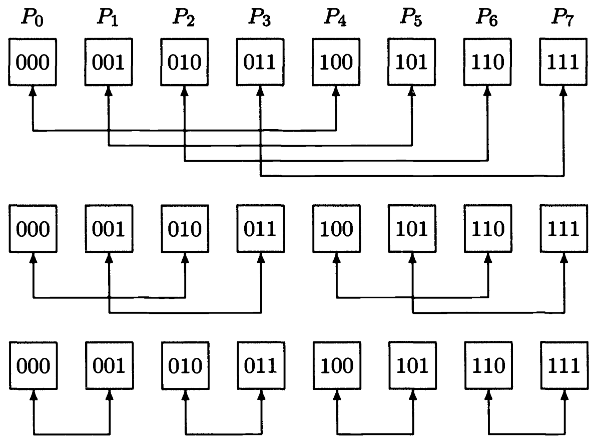


FIG. 5. *The d communication steps in the subcube-doubling algorithm* ($d = 3$).

cessors can maintain data proximity so that the same communication pattern may be followed by all processors throughout the computation. (By redundant update we mean that multiple processors perform the same computation on the same set of data and produce redundant results.) This paradigm is used in both algorithms we present in this paper. In addition, we show in the second algorithm how the aspect ratio of the subcube-grid may be chosen to reduce the computational cost as well as the communication cost of the parallel algorithm.

3. Algorithm I.

The first algorithm we propose is based on the Givens sequence illustrated in Fig. 6 by a 16×8 matrix. That is, the nonzero elements below the main diagonal of an m-by-n matrix are eliminated column by column in the order indicated in Fig. 6, where the (i, k) entries to be zeroed in the kth elimination stage are labeled by the integer k. In the parallel algorithm, the p processors cooperate to annihilate the $m - k$ nonzero elements in column k during the kth elimination stage. If a Givens rotation is applied to the ith row and the jth row to annihilate the leading nonzero $a_{j,k}$ in the jth row, then row i is referred to as the "pivot row."

As usual with parallel algorithms, we would like to achieve a balanced distribution of work load and a low volume of data movement and communication. A uniform work load distribution and a low communication cost contribute directly to the *speedup*, which is the ultimate goal of a concurrent algorithm.

3.1. Data mapping strategy.

Since there is no globally shared memory in the hypercube, the data must be distributed among the processing nodes in some way, typically by rows if the computation is row oriented, or by columns if the computation is column oriented. *In either case*, there is a decision to be made concerning the way in which the rows or columns are mapped onto the processors. For example, given an m-by-n matrix A and p processors $P_0, P_1, P_2, \cdots, P_{p-1}$, *block mapping* may be used, where the first m/p rows (or n/p columns) are assigned to processor P_0, the next m/p rows (or n/p columns) are assigned to processor P_1, and so on. Alternatively, *wrap mapping* may be used, where consecutive rows (or columns) are assigned to consecutive processors, with assignment "wrapping around" to processor P_0 after a row (or column) is assigned to processor P_{p-1}.

$$\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times \\
1 & \times & \times & \times & \times & \times & \times & \times \\
1 & 2 & \times & \times & \times & \times & \times & \times \\
1 & 2 & 3 & \times & \times & \times & \times & \times \\
1 & 2 & 3 & 4 & \times & \times & \times & \times \\
1 & 2 & 3 & 4 & 5 & \times & \times & \times \\
1 & 2 & 3 & 4 & 5 & 6 & \times & \times \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & \times \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8
\end{pmatrix}$$

FIG. 6. *Column-by-column Givens sequence.*

Discussion about various mapping strategies for matrix computations can be found in [7], [10], and [12]. For our purpose, it suffices to observe that if block mapping is used to assign rows or columns of A to the p processors, then processor P_0 will become idle when the first m/p rows or the first n/p columns of A do not need to be modified any more. The other $p-1$ processors could become idle one after another subsequently for the same reason. On the other hand, the rationale behind the wrap mapping is to assign the data to the p processors in such a manner that every processor will be doing approximately the same amount of computation and communication throughout the entire process except for the last p steps. Although the p processors will become idle one after another in the last p steps, the idle time so incurred will not be significant if the work involved for each of these p steps is only a tiny fraction of the total work.

For Algorithm I, we allocate the m rows to the p processors using a wrap mapping. Although wrap mapping is not essential for the correctness of the algorithms we propose, it is important for efficiency because the latter depends very much on whether a balanced work load distribution can be maintained throughout the computing process. Figure 7 illustrates the wrap mapping of 16 rows to four processors.

3.2. The algorithm. For a given $m \times n$ matrix, Algorithm I has n stages, each stage consisting of nine steps. The nine steps of the kth stage are devised to annihilate the $m-k$ nonzero elements in locations $(k+1,k)$, $(k+2,k)$, $\cdots$, and (m,k). Steps $1-9$ of the kth stage of the algorithm can be divided into two distinct phases, namely an Independent Annihilation Phase (IAP) and a Cooperative Merging Phase (CMP). The IAP corresponds to Step 1, where each processor operates on its assigned data without communicating with other processors in the network. The CMP corresponds to Steps $2-9$, where it is necessary for each processor to exchange data with its d neighbors if a hypercube of dimension d is employed. To follow the step-by-step description, it is helpful to explicitly relate Steps $2-9$ to the underlining subcube-doubling communication scheme as below.

```
$\ell \leftarrow d$ (Step 2)
while $\ell > 0$ do
    send (my message) to processor with id different
    from my id in bit $b_{\ell-1}$. (Step 3)
receive a message (Step 3)
update (my message) as instructed. (Steps 4, 5)
$\ell \leftarrow \ell - 1$ (Step 6)
```

The complete algorithm includes **Steps 8** and **9** because actions different from Steps 4 and 5 must be taken when $\ell = 1$. Since Step 3 is identical for all n elimination stages, the same communication pattern is maintained by all processors throughout the computation. This is made possible by the new updating process described in detail in Steps 4, 5, and 9.

From our description of Steps 2 – 9 below, it will become apparent that in the CMP of the kth stage, the data to be exchanged between each pair of processors always consist of $(n - k + 1)$ floating-point numbers. It also turns out that the amount of computation performed by each processor after each exchange of data is the same.

We now describe the steps for the kth stage. An example is used along the way to help explain each of the nine steps. After the details for the kth stage are given, we shall use an example to demonstrate how the entire algorithm works and summarize the features of the communication algorithm we propose.

Step 1 (IAP) Among all of the rows with row number $i \geq k$, each processor uses the lowest numbered row as the pivot row to eliminate all of the off-diagonal nonzero elements in the kth column of its remaining rows by Givens rotations. Using the example in Fig. 7, the action of processor P_0 in the first elimination stage is illustrated in Fig. 8. The leading nonzeros in rows 5, 9, and 13 are annihilated by applying Givens rotations sequentially to the three pairs of rows, namely, {row 1, row 5}, {row 1, row 9}, and {row 1, row 13}.
Therefore, no communication is needed in the IAP. At the end of this step, every processor has one row with a nonzero in the kth column. We shall refer to this row as the "local pivot row." The (i, k) nonzero entries in these local pivot rows (except for element (k, k)) are to be annihilated at Steps 2 – 9 in the CMP. Figure 9 displays the remaining nonzero elements at the end of Step 1 for a 16×8 example when $p = 4$ and $k = 1$. The elements of each local pivot row are marked by its assigned processor P_i, $0 \leq i \leq 3$. The entries $(2,1)$, $(3,1)$, and $(4,1)$ are to be zeroed in the CMP.

Step 2 (CMP) $\ell \leftarrow d$, where d is the dimension of the hypercube.

Step 3 (CMP) Every processor sends its current local pivot row to the processor whose *id* differs in bit $b_{\ell-1}$. It also receives a row from the other processor. For the example in Fig. 9, when $k = 1$, $\ell = d = 2$, rows 1 and 3 are thereby made available to both P_0 and P_2, and rows 2 and 4 are thereby made available to both P_1 and P_3 at the end of this step. We shall denote this pair of rows as $\tilde{a}_{\rho_1,*}$ and $\tilde{a}_{\rho_2,*}$, where ρ_1 and ρ_2 are their respective row numbers in the matrix A, and $\rho_1 < \rho_2$.

Step 4 (CMP) Each processor computes a Givens rotation to eliminate the element $\tilde{a}_{\rho_2,k}$ by executing the following algorithm.

```
if $|\tilde{a}_{\rho_2,k}| \geq |\tilde{a}_{\rho_1,k}|$ then
    $t \leftarrow |\tilde{a}_{\rho_1,k}| / |\tilde{a}_{\rho_2,k}|$
    $s_{\rho_2,k} \leftarrow 1/\sqrt{1+t^2}$
    $c_{\rho_2,k} \leftarrow s_{\rho_2,k} \times t$
else
```

```
$t \leftarrow |\tilde{a}_{\rho_2,k}|/|\tilde{a}_{\rho_1,k}|$
$c_{\rho_2,k} \leftarrow 1/\sqrt{1+t^2}$
$s_{\rho_2,k} \leftarrow c_{\rho_2,k} \times t$
```

If row ρ_2 is originally assigned to this processor, then row ρ_1, row ρ_2, $c_{\rho_2,k}$, and $s_{\rho_2,k}$ are saved so that row ρ_2 can be updated in Step 9.
For the given example, when $k = 1$ and $\ell = d = 2$, P_2 saves row 1, row 3, $c_{3,1}$ and $s_{3,1}$, and P_3 saves row 2, row 4, $c_{4,1}$ and $s_{4,1}$.

Step 5 (CMP) Each processor "updates" row ρ_1 by executing the following algorithm. The updated row ρ_1 becomes the "current" local pivot row.

```
for $j = k, k+1, \ldots, n$ do
    $\tilde{a}_{\rho_1,j} \leftarrow c_{\rho_2,k}\tilde{a}_{\rho_1,j} + s_{\rho_2,k}\tilde{a}_{\rho_2,j}$
```

For the example in Fig. 9, when $k = 1$ and $\ell = 2$, row 1 is modified to become the current local pivot row in P_0 and P_2, whereas row 2 is modified to become the current local pivot row in P_1 and P_3. Note that redundant computation is performed.

Step 6 (CMP) $\ell \leftarrow \ell - 1$.

Step 7 (CMP) If $\ell > 1$ then go to Step 3.

Step 8 (CMP) Each processor sends its current local pivot row to the processor whose *id* differs in bit b_0.

Step 9 (CMP) The processor that was originally assigned row k executes the algorithms given in Step 4 and 5 to update row k. Each other processor modifies the row originally assigned to it by executing the following algorithm. (Note that this row is either the higher numbered row currently received or the one saved at Step 4.)

```
for $j = k, k+1, \ldots, n$ do
    $\tilde{a}_{\rho_2,j} \leftarrow c_{\rho_2,k}\tilde{a}_{\rho_1,j} - s_{\rho_2,k}\tilde{a}_{\rho_2,j}$
```

For the given example, when $k = 1$ and $\ell = 1$, processors P_0, P_1, P_2, and P_3 will modify, respectively, rows 1, 2, 3, and 4 simultaneously in this step.

3.3. An example. To demonstrate how Steps 2 – 9 in the CMP work in general, we trace the pair of rows (ρ_1, ρ_2), which are the data each processor works with in Steps 4, 5, and 9, for three elimination stages in Figs. 10–12, where a hypercube of dimension 3 is employed. Recall that ρ_1 and ρ_2 refer to the respective row numbers with $\rho_1 < \rho_2$. While the computation performed by Steps 4 and 5 updates only row ρ_1 regardless of its origin, the updating of row ρ_2 will be delayed and performed only by one processor in Step 9. It is important to note that the mapping between the actual row numbers to (ρ_1, ρ_2) changes with each communication step. For example, row 3 is referred to as row ρ_1 in the first column of Fig. 10 when pairing with row 7; and the updated row 3 is referred to as row ρ_2 in the second column when pairing with the updated row 1. Since row 3 was initially mapped to processor P_2, the current row 1 and row 3 (both of them have already been updated once) must be saved by P_2 so that row 3 can be properly updated in Step 9. To identify the row and the processor associated with the updating operation in Step 9, we have quoted the row number 3 in the (1,'3') pair owned by P_2 in the second column of Fig. 10. Note that each processor has exactly one quoted row to be updated in Step 9 during each elimination

$$\begin{pmatrix}
P_0 & P_0 & P_0 & P_0 & P_0 & P_0 & P_0 & P_0 \\
P_1 & P_1 & P_1 & P_1 & P_1 & P_1 & P_1 & P_1 \\
P_2 & P_2 & P_2 & P_2 & P_2 & P_2 & P_2 & P_2 \\
P_3 & P_3 & P_3 & P_3 & P_3 & P_3 & P_3 & P_3 \\
P_0 & P_0 & P_0 & P_0 & P_0 & P_0 & P_0 & P_0 \\
P_1 & P_1 & P_1 & P_1 & P_1 & P_1 & P_1 & P_1 \\
P_2 & P_2 & P_2 & P_2 & P_2 & P_2 & P_2 & P_2 \\
P_3 & P_3 & P_3 & P_3 & P_3 & P_3 & P_3 & P_3 \\
P_0 & P_0 & P_0 & P_0 & P_0 & P_0 & P_0 & P_0 \\
P_1 & P_1 & P_1 & P_1 & P_1 & P_1 & P_1 & P_1 \\
P_2 & P_2 & P_2 & P_2 & P_2 & P_2 & P_2 & P_2 \\
P_3 & P_3 & P_3 & P_3 & P_3 & P_3 & P_3 & P_3 \\
P_0 & P_0 & P_0 & P_0 & P_0 & P_0 & P_0 & P_0 \\
P_1 & P_1 & P_1 & P_1 & P_1 & P_1 & P_1 & P_1 \\
P_2 & P_2 & P_2 & P_2 & P_2 & P_2 & P_2 & P_2 \\
P_3 & P_3 & P_3 & P_3 & P_3 & P_3 & P_3 & P_3
\end{pmatrix}$$

FIG. 7. *Wrap mapping of* 16 *rows to four processors.*

$$\begin{pmatrix}
a_{1,1} & a_{1,2} & \cdots & a_{1,8} \\
a_{5,1} & a_{5,2} & \cdots & a_{5,8} \\
a_{9,1} & a_{9,2} & \cdots & a_{9,8} \\
a_{13,1} & a_{13,2} & \cdots & a_{13,8}
\end{pmatrix}
\longrightarrow
\begin{pmatrix}
\tilde{a}_{1,1} & \tilde{a}_{1,2} & \cdots & \tilde{a}_{1,8} \\
0 & \tilde{a}_{5,2} & \cdots & \tilde{a}_{5,8} \\
0 & \tilde{a}_{9,2} & \cdots & \tilde{a}_{9,8} \\
0 & \tilde{a}_{13,2} & \cdots & \tilde{a}_{13,8}
\end{pmatrix}$$

FIG. 8. *The action of P_0 in the first elimination stage.*

$$\begin{pmatrix}
P_0 & P_0 & P_0 & P_0 & P_0 & P_0 & P_0 & P_0 \\
P_1 & P_1 & P_1 & P_1 & P_1 & P_1 & P_1 & P_1 \\
P_2 & P_2 & P_2 & P_2 & P_2 & P_2 & P_2 & P_2 \\
P_3 & P_3 & P_3 & P_3 & P_3 & P_3 & P_3 & P_3 \\
0 & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times
\end{pmatrix}$$

FIG. 9. *The distribution of local pivot rows.*

stage.

Processors	$\ell = 3$ (ρ_1, ρ_2)	$\ell = 2$ (ρ_1, ρ_2)	$\ell = 1$ (ρ_1, ρ_2)	Step 9 ρ_1 or ρ_2
P_0	(1, 5)	(1, 3)	('1', 2)	'1'
P_1	(2, 6)	(2, 4)	(1 ,'2')	'2'
P_2	(3, 7)	(1,'3')	(1 , 2)	'3'
P_3	(4, 8)	(2,'4')	(1 , 2)	'4'
P_4	(1,'5')	(1, 3)	(1 , 2)	'5'
P_5	(2,'6')	(2, 4)	(1 , 2)	'6'
P_6	(3,'7')	(1, 3)	(1 , 2)	'7'
P_7	(4,'8')	(2, 4)	(1 , 2)	'8'

FIG. 10. *The first elimination stage.*

Processors	$\ell = 3$ (ρ_1, ρ_2)	$\ell = 2$ (ρ_1, ρ_2)	$\ell = 1$ (ρ_1, ρ_2)	Step 9 ρ_1 or ρ_2
P_0	(5,'9')	(3, 5)	(2 , 3)	'9'
P_1	(2, 6)	(2, 4)	('2', 3)	'2'
P_2	(3, 7)	(3, 5)	(2 ,'3')	'3'
P_3	(4, 8)	(2,'4')	(2 , 3)	'4'
P_4	(5, 9)	(3,'5')	(2 , 3)	'5'
P_5	(2,'6')	(2, 4)	(2 , 3)	'6'
P_6	(3,'7')	(3, 5)	(2 , 3)	'7'
P_7	(4,'8')	(2, 4)	(2 , 3)	'8'

FIG. 11. *The second elimination stage.*

Processors	$\ell = 3$ (ρ_1, ρ_2)	$\ell = 2$ (ρ_1, ρ_2)	$\ell = 1$ (ρ_1, ρ_2)	Step 9 ρ_1 or ρ_2
P_0	(5, '9')	(3, 5)	(3 , 4)	'9'
P_1	(6,'10')	(4, 6)	(3 , 4)	'10'
P_2	(3, 7)	(3, 5)	('3', 4)	'3'
P_3	(4, 8)	(4, 6)	(3 ,'4')	'4'
P_4	(5, 9)	(3,'5')	(3 , 4)	'5'
P_5	(6, 10)	(4,'6')	(3 , 4)	'6'
P_6	(3, '7')	(3, 5)	(3 , 4)	'7'
P_7	(4, '8')	(4, 6)	(3 , 4)	'8'

FIG. 12. *The third elimination stage.*

3.4. A summary of important features. We summarize the features of the proposed communication algorithm below. First, given a hypercube of dimension d, each processor has d neighbors and the communication algorithm executed at each elimination stage involves exactly d steps. At each of the d steps, each processor in the hypercube exchanges a message of the same length with a neighboring processor. Since the $p/2$ exchanges at each step involve $p/2$ distinct pairs of processors and use $p/2$ separate communication channels, they can occur *simultaneously*.

Second, each processor communicates with all of its d neighboring processors in *exactly the same order* at every elimination stage. For example, in Figs. 10 – 12, processor P_0 always communicates with processors P_4, then P_2, and last P_1.

Third, it is clear from our description of the algorithm and from Figs. 10 – 12 that some rows are *redundantly* updated by more than one processor concurrently. For example, in Fig. 10 when $\ell = 3$, row 1 is redundantly updated by processor P_4, and when $\ell = 2$, row 1 is redundantly updated by processors P_2, P_4, and P_6. The redundant computation allows us to use the same communication algorithm for every elimination stage regardless of whether that processor owns the lowest numbered row. For example, in Fig. 11, among all the rows with row number $i \geq 2$, the second row of A is the lowest numbered row that needs to be modified last and the most number of times. Although the lowest numbered row is now located in processor P_1 instead of processor P_0 as in the first elimination stage, the same communication algorithm combined with the redundant computation makes sure that row 2 is properly updated, as are the other rows. In addition, the data each processor needs are always located in its neighboring processor in the hypercube.

Fourth, the algorithm has simple implementation. Note that when $k > n-p+1$, the processors will run out of rows one by one. Because the above algorithm requires all processors to participate in maintaining data proximity in the remaining stages, a uniform treatment of all cases can be achieved by simply assigning dummy data to processors who would otherwise finish earlier. With this trick the algorithm we described for the kth elimination stage can remain the same for $1 \leq k \leq n$.

Finally, the proposed communication algorithm can be easily generalized for Algorithm II, which we shall present in §5.

It is appropriate to point out that Algorithm I differs from the distributed Givens algorithms proposed by Chamberlain and Powell [3] and Pothen et al. [16], [17] in the communication algorithm we employ for merging the local pivot rows. According to the timing results to be discussed in §7, our idea of using otherwise idle processors to perform redundant computation appears to be effective in keeping the communication algorithm simple, efficient, and versatile for use in a generalized version of Algorithm I and other algorithms.

4. Performance analysis of Algorithm I.

In this section we analyze the performance of Algorithm I in factoring an $m \times n$ matrix using a hypercube of dimension d. Letting p denote the total number of processors, we have $p = 2^d$. For convenience we assume that m and n are integral multiples of p. In our analysis we assume that the time required for a multiplicative floating-point operation is τ, and that the time required to transmit η floating-point numbers from one processor to a neighboring processor is $\eta\lambda + \beta$, where β is the start-up time for sending a message and λ is the time needed to transmit a floating-point number across one link between adjacent processors in the hypercube.

We shall compare the performance of Algorithm I with the sequential Givens algorithm, for which the total arithmetic cost is given by

$$(1) \qquad \begin{aligned} T_s(m,n) &= 4\tau\sum_{k=1}^{n}(m-k)(n-k+1) \\ &= \frac{2n^2(3m-n)}{3}\tau + 2mn\tau - 2n^2\tau - \frac{4}{3}n\tau\,, \end{aligned}$$

when $m \geq n$, and

$$\tilde{T}_s(m,n) = 4\tau \sum_{k=1}^{m} (m-k)(n-k+1) = \frac{2m^2(3n-m)}{3}\tau - 2mn\tau + 2m^2\tau - \frac{4}{3}m\tau\,, \tag{2}$$

when $m \leq n$.

Note that the wrap mapping of rows of the matrix to the p processors dictates that the size of the largest submatrix in an individual processor is $(m/p) \times (n-k+1)$ for the kth elimination stage when $k = 1, 2, \cdots, p$, and $(m/p - 1) \times (n-k+1)$ when $k = p+1, p+2, \cdots, 2p$, and so on. When $m \geq n$, the arithmetic cost of Algorithm I is thus given by

$$\begin{aligned} T_I^A(m,n,p) &= 4\tau \sum_{k=1}^{n/p} \sum_{j=1}^{p} \left(\frac{m}{p} - k\right)(n - p(k-1) - j + 1) \\ &\quad + 2\tau \sum_{k=1}^{n} d(n-k+1) \\ &= \frac{2n^2(3m-n)}{3p}\tau + n^2 d\tau - n^2\tau + \left(\frac{2mn - n^2}{p}\right)\tau \\ &\quad + O(np)\,, \end{aligned} \tag{3}$$

where $d = \log_2 p$. For each of the n elimination stages, d "nearest-neighbor" exchanges are required, involving in the kth stage a row of size $(n-k+1)$. The communication cost is therefore given by

$$\begin{aligned} T_I^C(n,p) &= \beta \sum_{k=1}^{n} 2d + \lambda \sum_{k=1}^{n} 2d(n-k+1) \\ &= 2nd\beta + \left(n^2 + n\right) d\lambda. \end{aligned} \tag{4}$$

When $m \leq n$, we have

$$\begin{aligned} \tilde{T}_I^A(m,n,p) &= 4\tau \sum_{k=1}^{m/p} \sum_{j=1}^{p} \left(\frac{m}{p} - k\right)(n - p(k-1) - j + 1) \\ &\quad + 2\tau \sum_{k=1}^{m} d(n-k+1) \\ &= \frac{2m^2(3n-m)}{3p}\tau + \left(2mn - m^2\right) d\tau - \left(2mn - m^2\right)\tau \\ &\quad + \frac{m^2}{p}\tau + O(mp) \end{aligned} \tag{5}$$

and

$$\begin{aligned} \tilde{T}_I^C(m,n,p) &= \beta \sum_{k=1}^{m} 2d + \lambda \sum_{k=1}^{m} 2d(n-k+1) \\ &= 2md\beta + \left(2mn - m^2 + m\right) d\lambda\,. \end{aligned} \tag{6}$$

When $m = n$, we can further simplify equations (3), (4), (5), and (6) as

$$\begin{aligned} T_I^A(n,n,p) &= \tilde{T}_I^A(n,n,p) \\ &= \frac{4n^3}{3p}\tau + n^2 d\tau - n^2\tau + \frac{n^2}{p}\tau + O(np) \end{aligned} \tag{7}$$

and

$$\begin{aligned} T_I^C(n,p) &= \tilde{T}_I^C(n,n,p) \\ &= 2nd\beta + n^2 d\lambda + nd\lambda\,. \end{aligned} \tag{8}$$

Comparing the parallel time $\left(T_I^A + T_I^C\right)$ with the optimal time (T_s/p) for the case $m \geq n$, and $(\tilde{T}_I^A + \tilde{T}_I^C)$ with the optimal time $(\tilde{T}_s/p)$ for the case $m \leq n$, we conclude that Algorithm I is optimal in its leading term.

5. Algorithm II.

Algorithm II configures the hypercube as a $\gamma_1 \times \gamma_2$ subcube-grid, where the aspect ratio will be chosen to optimize the performance. Such a mapping allows us to apply the hypercube communication algorithm employed in Algorithm I to each subset of processors that form either a row or a column in the subcube-grid. From now on, we shall refer to the processor in the (i,j) position of the grid by $P(i,j)$ or by its *id*. We shall use $P(i,*)$ to denote the subcube consisting of the processors assigned to the ith row of the grid, and $P(*,j)$ to denote the subcube consisting of the processors assigned to the jth column of the grid.

Algorithm II is based on the same Givens sequence employed in Algorithm I. In Algorithm II, the data mapping strategy can be understood as wrapping the rows of the $m \times n$ matrix around the γ_1 subcubes, namely, $P(1,*)$, $P(2,*)$, $\cdots$, and $P(\gamma_1,*)$. Within each subcube $P(i,*)$, the elements of each allocated row are wrapped around the γ_2 processors according to their column subscripts. Figure 13 illustrates this data mapping strategy for a 16×16 matrix on the 4×4 subcube-grid given in Fig. 3.

$$\begin{pmatrix}
p_0 & p_1 & p_2 & p_3 & p_0 & p_1 & p_2 & p_3 & p_0 & p_1 & p_2 & p_3 & p_0 & p_1 & p_2 & p_3 \\
p_4 & p_5 & p_6 & p_7 & p_4 & p_5 & p_6 & p_7 & p_4 & p_5 & p_6 & p_7 & p_4 & p_5 & p_6 & p_7 \\
p_8 & p_9 & p_{10} & p_{11} & p_8 & p_9 & p_{10} & p_{11} & p_8 & p_9 & p_{10} & p_{11} & p_8 & p_9 & p_{10} & p_{11} \\
p_{12} & p_{13} & p_{14} & p_{15} & p_{12} & p_{13} & p_{14} & p_{15} & p_{12} & p_{13} & p_{14} & p_{15} & p_{12} & p_{13} & p_{14} & p_{15} \\
p_0 & p_1 & p_2 & p_3 & p_0 & p_1 & p_2 & p_3 & p_0 & p_1 & p_2 & p_3 & p_0 & p_1 & p_2 & p_3 \\
p_4 & p_5 & p_6 & p_7 & p_4 & p_5 & p_6 & p_7 & p_4 & p_5 & p_6 & p_7 & p_4 & p_5 & p_6 & p_7 \\
p_8 & p_9 & p_{10} & p_{11} & p_8 & p_9 & p_{10} & p_{11} & p_8 & p_9 & p_{10} & p_{11} & p_8 & p_9 & p_{10} & p_{11} \\
p_{12} & p_{13} & p_{14} & p_{15} & p_{12} & p_{13} & p_{14} & p_{15} & p_{12} & p_{13} & p_{14} & p_{15} & p_{12} & p_{13} & p_{14} & p_{15} \\
p_0 & p_1 & p_2 & p_3 & p_0 & p_1 & p_2 & p_3 & p_0 & p_1 & p_2 & p_3 & p_0 & p_1 & p_2 & p_3 \\
p_4 & p_5 & p_6 & p_7 & p_4 & p_5 & p_6 & p_7 & p_4 & p_5 & p_6 & p_7 & p_4 & p_5 & p_6 & p_7 \\
p_8 & p_9 & p_{10} & p_{11} & p_8 & p_9 & p_{10} & p_{11} & p_8 & p_9 & p_{10} & p_{11} & p_8 & p_9 & p_{10} & p_{11} \\
p_{12} & p_{13} & p_{14} & p_{15} & p_{12} & p_{13} & p_{14} & p_{15} & p_{12} & p_{13} & p_{14} & p_{15} & p_{12} & p_{13} & p_{14} & p_{15} \\
p_0 & p_1 & p_2 & p_3 & p_0 & p_1 & p_2 & p_3 & p_0 & p_1 & p_2 & p_3 & p_0 & p_1 & p_2 & p_3 \\
p_4 & p_5 & p_6 & p_7 & p_4 & p_5 & p_6 & p_7 & p_4 & p_5 & p_6 & p_7 & p_4 & p_5 & p_6 & p_7 \\
p_8 & p_9 & p_{10} & p_{11} & p_8 & p_9 & p_{10} & p_{11} & p_8 & p_9 & p_{10} & p_{11} & p_8 & p_9 & p_{10} & p_{11} \\
p_{12} & p_{13} & p_{14} & p_{15} & p_{12} & p_{13} & p_{14} & p_{15} & p_{12} & p_{13} & p_{14} & p_{15} & p_{12} & p_{13} & p_{14} & p_{15}
\end{pmatrix}$$

FIG. 13. *The wrap mapping of a* 16×16 *matrix to the* 4×4 *subcube-grid in Fig.* 3.

Algorithm II has n stages, each consisting of two phases. In the Independent Annihilation Phase (IAP) of the kth elimination stage, each subcube $P(i,*)$ independently annihilates the nonzero elements below the main diagonal in the kth column using its lowest numbered row as the local pivot row. The algorithm for the IAP requires no communication between the processors in different subcubes $P(i_1,*)$ and $P(i_2,*)$, where $i_1 \neq i_2$, whereas the processors consisting of each subcube $P(i,*)$ need to communicate among themselves during the IAP. For the example in this section we employ the 4×4 subcube-grid given in Fig. 3. Figure 14 displays the data assigned to the subcube $P(1,*)$, which is a submatrix consisting of rows 1, 5, 9, and 13 of a 16×16 matrix A. Within the subcube, the columns of the assigned submatrix are wrapped around the γ_2 processors of $P(1,*)$. Letting $a_{i,j}$ denote the (i,j) element of matrix A, Figs. 15, 16, 17, and 18 display the data assigned to processors $P(1,1)$, $P(1,2)$, $P(1,3)$, and $P(1,4)$, respectively.

$$\begin{pmatrix}
P_0 & P_1 & P_2 & P_3 & P_0 & P_1 & P_2 & P_3 & P_0 & P_1 & P_2 & P_3 & P_0 & P_1 & P_2 & P_3 \\
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
P_0 & P_1 & P_2 & P_3 & P_0 & P_1 & P_2 & P_3 & P_0 & P_1 & P_2 & P_3 & P_0 & P_1 & P_2 & P_3 \\
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
P_0 & P_1 & P_2 & P_3 & P_0 & P_1 & P_2 & P_3 & P_0 & P_1 & P_2 & P_3 & P_0 & P_1 & P_2 & P_3 \\
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
P_0 & P_1 & P_2 & P_3 & P_0 & P_1 & P_2 & P_3 & P_0 & P_1 & P_2 & P_3 & P_0 & P_1 & P_2 & P_3 \\
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times
\end{pmatrix}$$

FIG. 14. *The wrap mapping of the submatrix within subcube $P(1,*)$.*

$$\begin{pmatrix}
a_{1,1} & a_{1,5} & a_{1,9} & a_{1,13} \\
a_{5,1} & a_{5,5} & a_{5,9} & a_{5,13} \\
a_{9,1} & a_{9,5} & a_{9,9} & a_{9,13} \\
a_{13,1} & a_{13,5} & a_{13,9} & a_{13,13}
\end{pmatrix}$$

FIG. 15. *Data assigned to processor P_0.*

$$\begin{pmatrix}
a_{1,2} & a_{1,6} & a_{1,10} & a_{1,14} \\
a_{5,2} & a_{5,6} & a_{5,10} & a_{5,14} \\
a_{9,2} & a_{9,6} & a_{9,10} & a_{9,14} \\
a_{13,2} & a_{13,6} & a_{13,10} & a_{13,14}
\end{pmatrix}$$

FIG. 16. *Data assigned to processor P_1.*

To eliminate the nonzeros $a_{5,1}, a_{9,1}$, and $a_{13,1}$, processor P_0 computes the multiplier pairs $\{c_{\rho,1}, s_{\rho,1}\}$ for $\rho = 5, 9, 13$ and updates the "local pivot element," $a_{1,1}$, by the following algorithm.

```
if $|a_{\rho,1}| \geq |a_{1,1}|$ then
    $t \leftarrow |a_{1,1}|/|a_{\rho,1}|$
    $s_{\rho,1} \leftarrow 1/\sqrt{1+t^2}$
    $c_{\rho,1} \leftarrow s_{\rho,1} \times t$
    $a_{1,1} \leftarrow |a_{1,1}|\sqrt{1+t^2}$
else
    $t \leftarrow |a_{\rho,1}|/|a_{1,1}|$
    $c_{\rho,1} \leftarrow 1/\sqrt{1+t^2}$
    $s_{\rho,1} \leftarrow c_{\rho,1} \times t$
    $a_{1,1} \leftarrow |a_{1,1}|\sqrt{1+t^2}$
```

$$\begin{pmatrix} a_{1,3} & a_{1,7} & a_{1,11} & a_{1,15} \\ a_{5,3} & a_{5,7} & a_{5,11} & a_{5,15} \\ a_{9,3} & a_{9,7} & a_{9,11} & a_{9,15} \\ a_{13,3} & a_{13,7} & a_{13,11} & a_{13,15} \end{pmatrix}$$

FIG. 17. *Data assigned to processor P_2.*

$$\begin{pmatrix} a_{1,4} & a_{1,8} & a_{1,12} & a_{1,16} \\ a_{5,4} & a_{5,8} & a_{5,12} & a_{5,16} \\ a_{9,4} & a_{9,8} & a_{9,12} & a_{9,16} \\ a_{13,4} & a_{13,8} & a_{13,12} & a_{13,16} \end{pmatrix}$$

FIG. 18. *Data assigned to processor P_3.*

A message containing the "updated" $a_{1,1}$ and *all of the computed multiplier pairs* will then be made available to every processor in the subcube $P(1,*)$ using the following recursive exchange algorithm. Recall that the *id*'s of processors in each subcube $P(i,*)$ differ only in their rightmost d_2 bits. To broadcast the multiplier pairs and the local pivot element to all processors within each subcube $P(i,*)$, we modify the basic subcube-doubling communication algorithm in the following manner. The processor who has the pivot column will compose a message consisting of the computed multipliers and the local pivot element, whereas the other processors will simply prepare a dummy message. The modified algorithm works as follows.

```
$\ell \leftarrow d_2$
**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) is (dummy message)
        (my message) $\leftarrow$ (received message)
    $\ell \leftarrow \ell - 1$
```

The subcubes $P(1,*)$, $P(2,*)$, $P(3,*)$, and $P(4,*)$ each perform essentially the same IAP task with respect to their own data independently and simultaneously. The communication channels employed by the four subcubes are displayed in Fig. 19. Therefore, after d_2 exchanges within each subcube, all processors will have the multipliers they need to update the remaining elements independently. The resulting matrix has at most γ_1 rows with a nonzero in the kth column. For $k = 1$, Fig. 20 illustrates the mapping of the γ_1 local pivot rows at the end of the IAP.

The remaining off-diagonal nonzero elements in the first column can now be eliminated by the γ_2 subcubes $P(*,j)$, $j = 1, 2, \cdots, \gamma_2$, independently and simultaneously.

Figure 21 displays the data to be affected in the subcube $P(*,1) = \{P_0, P_4, P_8, P_{12}\}$.

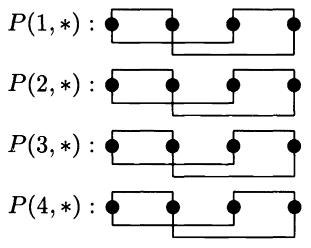


FIG. 19. *The communication channels employed by the subcubes during the* IAP.

$$\begin{pmatrix}
P_0 & P_1 & P_2 & P_3 & P_0 & P_1 & P_2 & P_3 & P_0 & P_1 & P_2 & P_3 & P_0 & P_1 & P_2 & P_3 \\
P_4 & P_5 & P_6 & P_7 & P_4 & P_5 & P_6 & P_7 & P_4 & P_5 & P_6 & P_7 & P_4 & P_5 & P_6 & P_7 \\
P_8 & P_9 & P_{10} & P_{11} & P_8 & P_9 & P_{10} & P_{11} & P_8 & P_9 & P_{10} & P_{11} & P_8 & P_9 & P_{10} & P_{11} \\
P_{12} & P_{13} & P_{14} & P_{15} & P_{12} & P_{13} & P_{14} & P_{15} & P_{12} & P_{13} & P_{14} & P_{15} & P_{12} & P_{13} & P_{14} & P_{15} \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times
\end{pmatrix}$$

FIG. 20. *End of the* IAP *during the first elimination stage.*

$$\begin{pmatrix}
P_0 & \times & \times & \times & P_0 & \times & \times & \times & P_0 & \times & \times & \times & P_0 & \times & \times & \times \\
P_4 & \times & \times & \times & P_4 & \times & \times & \times & P_4 & \times & \times & \times & P_4 & \times & \times & \times \\
P_8 & \times & \times & \times & P_8 & \times & \times & \times & P_8 & \times & \times & \times & P_8 & \times & \times & \times \\
P_{12} & \times & \times & \times & P_{12} & \times & \times & \times & P_{12} & \times & \times & \times & P_{12} & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times
\end{pmatrix}$$

FIG. 21. *Data distribution in subcube* $P(*,1)$.

Comparing Fig. 21 with Fig. 9, we clearly see that the task to be performed by the subcube $P(*,1)$ is essentially Steps 2 – 9 of Algorithm I. Of course, the data now refers to the submatrix in Fig. 22, and the subcube $P(*,1)$ is of dimension $d/2$, and the *id*'s of the processors in this subcube differ only in their *leftmost* $d/2$ bits. It is straightforward to modify Steps 2 – 9 of Algorithm I to reflect these differences. Recall that during the IAP, each updated local pivot element was sent to all processors in the respective subcube together with the multipliers. When $k = 1$, $\{a_{1,1}, a_{2,1}, a_{3,1}, a_{4,1}\}$ are thus available in subcube $P(*,j)$ for all j. Figure 23 displays the data distribution in the subcube $P(*,2)$. Note that each processor in the subcube $P(*,2)$ has one more element in addition to the data originally assigned.

Comparing Fig. 23 with Fig. 9, it is clear that the task to be performed by the subcube $P(*, 2)$ is again essentially the same as described in Steps 2 – 9 of Algorithm I. The data now refers to the matrix in Fig. 24.

$$\begin{pmatrix} a_{1,1} & a_{1,5} & a_{1,9} & a_{1,13} \\ a_{2,1} & a_{2,5} & a_{2,9} & a_{2,13} \\ a_{3,1} & a_{3,5} & a_{3,9} & a_{3,13} \\ a_{4,1} & a_{4,5} & a_{4,9} & a_{4,13} \end{pmatrix}$$

FIG. 22. *Data to be processed by subcube $P(*, 1)$.*

$$\begin{pmatrix}
P_1 & P_1 & \times & \times & \times & P_1 & \times & \times & \times & P_1 & \times & \times & \times & P_1 & \times & \times \\
P_5 & P_5 & \times & \times & \times & P_5 & \times & \times & \times & P_5 & \times & \times & \times & P_5 & \times & \times \\
P_9 & P_9 & \times & \times & \times & P_9 & \times & \times & \times & P_9 & \times & \times & \times & P_9 & \times & \times \\
P_{13} & P_{13} & \times & \times & \times & P_{13} & \times & \times & \times & P_{13} & \times & \times & \times & P_{13} & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times
\end{pmatrix}$$

FIG. 23. *Data distribution in subcube $P(*, 2)$.*

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,6} & a_{1,10} & a_{1,14} \\ a_{2,1} & a_{2,2} & a_{2,6} & a_{2,10} & a_{2,14} \\ a_{3,1} & a_{3,2} & a_{3,6} & a_{3,10} & a_{3,14} \\ a_{4,1} & a_{4,2} & a_{4,6} & a_{4,10} & a_{4,14} \end{pmatrix}$$

FIG. 24. *Data to be processed by subcube $P(*, 2)$.*

The elements $\{a_{1,1}, a_{2,1}, a_{3,1}, a_{4,1}\}$ are needed in Step 4 to compute the multipliers. Therefore, the strategy of sending the "updated" local pivot elements together with the multiplers in the IAP is the most economic way to make these elements available to the respective processors. Similar arguments apply to the subcubes $P(*, 3)$ and $P(*, 4)$. The communication channels employed by the four subcubes during the CMP are displayed in Fig. 25.

Figure 26 displays the remaining elements of A after $n - \gamma_2$ elimination stages, where $p = 16$ and $\gamma_2 = 4$ in this example. Since $(\gamma_1 + \gamma_2 - 2i - 1)$ more processors will become idle at the end of each of the last γ_2 elimination stages, where $0 \le i \le (\gamma_2 - 1)$, the idle time is proportional to the amount of work each processor is assigned for one stage. When $m = n$ and $\gamma_1 = \gamma_2$, each processor has exactly one element to be zeroed or modified in the last γ_2 elimination stages. The idle time thus remains constant regardless of the size of the matrix. When $m > n$, γ_1 processors would become idle after each of the last γ_2 elimination stages. Because the elements remaining in each processor after $n - \gamma_2$ stages is proportional to $(m - n)/\gamma_1$, the idle time grows linearly with m for fixed γ_1 if $(m - n) \gg n$. When $m < n$, because γ_2 processors would become idle following each of the last γ_1 stages, the idle time grows linearly with n for fixed γ_2 if

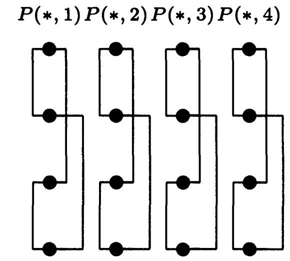


FIG. 25. *The communication channels employed by the subcubes during the* CMP.

$(n-m) \gg m$. As we shall see in the next section, γ_1 and γ_2 are to be chosen according to the shape of the matrix so that the performance of Algorithm II is optimized. The possibility of choosing γ_1 and γ_2 in proportion to m and n implies that the linear growth rate represents the worst case. Note that in the actual implementation of Algorithm II, the communication algorithms in both IAP and CMP phases require all processors to participate regardless of whether there is any computational work remaining for a particular processor. The idle time we mentioned above refers to the duration of time from the moment a processor has completely processed the assigned data of matrix A to the moment the parallel program ends. Thus the time such a processor spends working on dummy data is considered as idle time.

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

FIG. 26. *Data distribution for the last four elimination stages.*

6. Performance analysis of Algorithm II.

In this section we analyze the performance of Algorithm II in factoring an $m \times n$ matrix on a $\gamma_1 \times \gamma_2$ grid embedded in a hypercube of dimension d, where $d = d_1 + d_2$, $\gamma_1 = 2^{d_1}$ and $\gamma_2 = 2^{d_2}$. Letting p denote the total number of processors, we have $p = \gamma_1\gamma_2 = 2^d$. As before, we assume that m and n are integral multiples of p. The definitions for τ, β, and λ are as given in §4.

6.1. The case $m \geq n$.

When $m \geq n$, we first consider the case $\gamma_1 \geq \gamma_2$. To analyze the total arithmetic cost of Algorithm II, let us consider the kth elimination stage. During the IAP phase, the γ_1 subcubes $P(1,*)$, $P(2,*)$, $\cdots$, $P(\gamma_1,*)$ are performing essentially the same task on the γ_1 different submatrices independently and simultaneously. Within each subcube $P(i,*)$ the submatrix is further divided among the γ_2 processors consisting of the subcube. Letting $(\gamma_1/\gamma_2) = \alpha$, the wrap mapping of rows and columns of the matrix to the processor grid dictates that the size of the largest submatrix in an individual processor is $(m/\gamma_1) \times (n/\gamma_2)$ for $k = 1, 2, \cdots, \gamma_2$, $(m/\gamma_1) \times (n/\gamma_2 - 1)$ for $k = \gamma_2 + 1, \gamma_2 + 2, \cdots, 2\gamma_2$, $\cdots$, and $(m/\gamma_1 - 1) \times (n/\gamma_2 - \alpha)$

for $k = \gamma_1 + 1, \gamma_1 + 2, \cdots, \gamma_1 + \gamma_2$, and so on. The total arithmetic cost for the IAP is thus

$$\begin{aligned}(9)\quad T^A_{IAP}(m,n,\gamma_1,\gamma_2) &= 4\tau \sum_{k=1}^{n/\gamma_1} \sum_{j=1}^{\alpha} \sum_{i=1}^{\gamma_2} \left(\frac{m}{\gamma_1} - k\right)\left(\frac{n}{\gamma_2} - \alpha(k-1) - j + 1\right) \\ &= \frac{2n^2(3m-n)}{3p}\tau + \frac{2mn}{\gamma_1}\tau - n^2\left(\frac{\gamma_1+\gamma_2}{\gamma_1\gamma_2}\right)\tau \\ &\quad - n\tau - \frac{n}{3}\left(\frac{\gamma_1}{\gamma_2}\right)\tau \\ &= \frac{2n^2(3m-n)}{3p}\tau + \frac{2mn}{\gamma_1}\tau - n^2\left(\frac{\gamma_1+\gamma_2}{p}\right)\tau \\ &\quad + O(n).\end{aligned}$$

Recall that the multiplier pairs together with the updated local pivot element must be made available to the γ_2 processors within each subcube $P(i,*)$ using the recursive exchange algorithm. The total communication cost for the IAP is thus given by

$$\begin{aligned}(10)\qquad T^C_{IAP}(m,n,\gamma_1,\gamma_2) &= \beta\sum_{k=1}^{n} 2d_2 + \lambda \sum_{k=1}^{n/\gamma_1}\sum_{j=1}^{\gamma_1} 2d_2\left(\frac{m}{\gamma_1} - k + 1\right) \\ &= 2nd_2\beta + \left(\frac{2mn - n^2}{\gamma_1} + n\right) d_2\lambda .\end{aligned}$$

In the Cooperative Merging Phase (CMP) following the IAP, the γ_1 processors in each subcube $P(*,j)$, $1 \le j \le \gamma_2$, perform essentially Steps 2 – 9 of Algorithm I. When $k = 1$, every processor applies a Givens rotation d_1 times to a row of size at most $(n/\gamma_2 + 1)$, and exchanges a row of the same size with a neighboring processor d_1 times. Note that the longest row in an individual processor is $(n/\gamma_2 + 1)$ for $k = 1, 2, \cdots, \gamma_2$, and (n/γ_2) for $k = \gamma_2 + 1, \gamma_2 + 2, \cdots, 2\gamma_2$, and so on. We thus have

$$\begin{aligned}(11)\qquad T^A_{CMP}(n,\gamma_1,\gamma_2) &= 2\tau \sum_{k=1}^{n/\gamma_2}\sum_{j=1}^{\gamma_2} d_1\left(\frac{n}{\gamma_2} - k + 2\right) \\ &= \left(\frac{n^2}{\gamma_2} + 3n\right) d_1\tau\end{aligned}$$

and

$$\begin{aligned}(12)\qquad T^C_{CMP}(n,\gamma_1,\gamma_2) &= \beta\sum_{k=1}^{n} 2d_1 + \lambda\sum_{k=1}^{n/\gamma_2}\sum_{j=1}^{\gamma_2} 2d_1\left(\frac{n}{\gamma_2} - k + 2\right) \\ &= 2nd_1\beta + \left(\frac{n^2}{\gamma_2} + 3n\right) d_1\lambda .\end{aligned}$$

The total arithmetic and communication costs for Algorithm II are thus given by

$$\begin{aligned}(13)\qquad T^A_{II}(m,n,\gamma_1,\gamma_2) &= T^A_{IAP}(m,n,\gamma_1,\gamma_2) + T^A_{CMP}(n,\gamma_1,\gamma_2) \\ &= \frac{2n^2(3m-n)}{3p}\tau + \frac{n^2}{\gamma_2}d_1\tau + \frac{2mn}{\gamma_1}\tau \\ &\quad - n^2\left(\frac{\gamma_1+\gamma_2}{p}\right)\tau + O(nd_1)\end{aligned}$$

and

$$
\begin{aligned}
(14)\qquad T_{II}^{C}(m,n,\gamma_1,\gamma_2) &= T_{IAP}^{C}(m,n,\gamma_1,\gamma_2) + T_{CMP}^{C}(n,\gamma_1,\gamma_2) \\
&= 2nd\beta + \frac{2mn-n^2}{\gamma_1} d_2\lambda + \frac{n^2}{\gamma_2} d_1\lambda + O(nd)\,.
\end{aligned}
$$

Since $\gamma_1\gamma_2 = p$, the parallel time of Algorithm II can be expressed as a function of m, n, γ_1, and p as follows:

$$
\begin{aligned}
(15)\qquad T_{II}(m,n,\gamma_1,p) &= T_{II}^{A}(m,n,\gamma_1,\gamma_2) + T_{II}^{C}(m,n,\gamma_1,\gamma_2) \\
&= \frac{2n^2(3m-n)}{3p}\tau + \frac{2mn-n^2}{\gamma_1} d_2\lambda + \frac{n^2}{\gamma_2} d_1(\tau+\lambda) \\
&\quad + \frac{2mn}{\gamma_1}\tau - n^2\left(\frac{\gamma_1+\gamma_2}{p}\right)\tau + O(nd) \\
&= \frac{2n^2(3m-n)}{3p}\tau + \frac{2mn-n^2}{\gamma_1}(d-d_1)\lambda \\
&\quad + \frac{n^2}{p}\gamma_1 d_1(\tau+\lambda) + \frac{2mn}{\gamma_1}\tau - n^2\left(\frac{\gamma_1{}^2 p}{p\gamma_1}\right)\tau \\
&\quad + O(nd)\,.
\end{aligned}
$$

If $\gamma_1 \le \gamma_2$, T_{IAP}^{C}, T_{CMP}^{A}, and T_{CMP}^{C} remain the same as given by (10) – (12), whereas T_{IAP}^{A} is now computed by (16), where we use $\tilde{\alpha}$ to denote γ_2/γ_1:

$$
\begin{aligned}
(16)\ T_{IAP}^{A}(m,n,\gamma_1,\gamma_2) &= 4\tau \sum_{k=1}^{n/\gamma_2}\sum_{j=1}^{\tilde{\alpha}}\sum_{i=1}^{\gamma_1}\left(\frac{m}{\gamma_1} - \tilde{\alpha}(k-1) - j\right)\left(\frac{n}{\gamma_2} - k + 1\right) \\
&= \frac{2n^2(3m-n)}{3p}\tau + \frac{2mn}{\gamma_1}\tau - n^2\left(\frac{\gamma_1+\gamma_2}{\gamma_1\gamma_2}\right)\tau \\
&\quad - n\tau - \frac{n}{3}\left(\frac{\gamma_2}{\gamma_1}\right)\tau\,.
\end{aligned}
$$

Comparing (16) with (9), we see that they differ only in one of the low-order terms. Therefore, for $m \ge n$, we shall use (15) to compute $T_{II}(m,n,\gamma_1,p)$ for all values of γ_1. Note that our analysis of the communication cost, as summarized by (14), indicates that the total number of messages exchanged between each pair of processors is independent of the choice of γ_1. Accordingly, the contribution of *start-up* time to the communication cost of Algorithm II remains the same for all values of γ_1.

One objective of our analysis is to find the value of γ_1 that minimizes the cost of the parallel algorithm. To find γ_1, we set

$$
\frac{\partial T_{II}(m,n,\gamma_1,p)}{\partial \gamma_1} = 0\,,
$$

where $T_{II}(m,n,\gamma_1,p)$ is defined by (15). By noting that $d = \log_2 p$, $d_1 = \log_2 \gamma_1$, and $\partial d_1/\partial\gamma_1 = (\ln\gamma_1)/(\ln 2)$, we obtain the following result by setting $\partial T_{II}/\partial\gamma_1 = 0$:

$$
(17)\qquad f\left(p,\gamma_1,\frac{m}{n},\frac{\tau}{\lambda}\right) = a\left(\frac{\tau}{\lambda}+1\right)(\ln\gamma_1 + 1)\gamma_1{}^2 - \left(\frac{\tau}{\lambda}\right)\gamma_1{}^2
$$

$$\begin{aligned} &+ ap\left(2\frac{m}{n}-1\right)\ln\gamma_1 \\ &- p\left(\frac{\tau}{\lambda}+\log_2 p + a\right)\left(2\frac{m}{n}-1\right) \\ = \ &0, \end{aligned}$$

where $a = 1/(\ln 2)$.

Therefore, for fixed p, m/n, and τ/λ, the value of γ_1 that minimizes $T_{II}(m, n, \gamma_1, p)$ can be obtained by finding the solution to $f(p, \gamma_1, m/n, \tau/\lambda) = 0$, which is a nonlinear equation in the variable γ_1. Since γ_1 must be an integral power of 2, we choose $\gamma_1 = 2^k$ for $0 \leq k \leq \log_2 p$, with 2^k as close as possible to the solution of $f(p, \gamma_1, m/n, \tau/\lambda) = 0$. Although the "optimal" γ_1 chosen in this manner does not necessarily minimize $T_{II}(m, n, \gamma_1, p)$, the numerical experiments to be presented in §7 indicate that for each test problem, the execution time of Algorithm II using the chosen γ_1 either achieves or is very close to the actual minimum over all possible values of γ_1.

In order to see how the optimal γ_1 varies with the ratio τ/λ, in Table 1 we list the numerical solution to $f(p, \gamma_1, m/n, \tau/\lambda) = 0$ for different values of τ/λ when p and m/n remain fixed. The optimal γ_1's chosen based on these numerical solutions are displayed in Table 2. From Tables 1 and 2 we observe that the optimal $d_1 = \log_2 \gamma_1$, which is the optimal dimension of the subcube formed by each column of the grid, appears to be very insensitive to the ratio τ/λ.

TABLE 1
Numerical solution to $f(p, \gamma_1, m/n, \tau/\lambda) = 0$.

$m \geq n$		Different values of τ/λ						
p	m/n	1000	10	5	1	0.2	0.1	0.0
16	1.0	2.86	3.06	3.20	3.62	3.89	3.94	4.0
16	1.5	3.71	3.95	4.11	4.63	4.97	5.03	5.11
16	19.8	12.33	12.58	12.76	13.35	13.76	13.83	13.92
64	1.0	4.86	5.42	5.80	6.95	7.70	7.84	8.0
64	1.5	6.41	7.11	7.58	9.04	10.01	10.19	10.39
64	19.8	22.41	23.79	24.78	27.96	30.13	30.54	31.00
128	1.0	6.42	7.30	7.89	9.68	10.85	11.06	11.30
128	1.5	8.53	9.64	10.38	12.67	14.19	14.47	14.79
128	19.8	30.38	32.87	34.62	40.24	44.08	44.81	45.63
256	1.0	8.53	9.89	10.78	13.50	15.29	15.62	18.54
256	1.5	11.41	13.13	14.27	17.78	20.11	20.55	21.04
256	19.8	41.30	45.52	48.44	57.77	64.16	65.37	66.74
1024	1.0	15.34	18.41	20.40	26.41	30.40	31.15	32.00
1024	1.5	20.69	24.64	27.23	35.10	40.35	41.35	42.47
1024	19.8	76.92	87.78	95.15	118.41	134.34	137.39	140.82

6.2. The case $m \leq n$. Similarly, when $m \leq n$, we have the cases $\gamma_1 \geq \gamma_2$ and $\gamma_1 \leq \gamma_2$. Although our derivation below is for the case $\gamma_1 \leq \gamma_2$, $\tilde{T}^A_{IAP}$ is different only in one of the low-order terms when $\gamma_1 \geq \gamma_2$ and $\tilde{T}^C_{IAP}$, $\tilde{T}^A_{CMP}$, and $\tilde{T}^C_{CMP}$ remain unchanged. We shall therefore use the following formula for all values of γ_1. Letting

TABLE 2
Predicted optimal γ_1 when $m \geq n$.

$m \geq n$		Different values of τ/λ						
p	m/n	1000	10	5	1	0.2	0.1	0
16	1.0	2	4	4	4	4	4	4
16	1.5	4	4	4	4	4	4	4
16	19.8	16	16	16	16	16	16	16
64	1.0	4	4	4	8	8	8	8
64	1.5	8	8	8	8	8	8	8
64	19.8	16	16	32	32	32	32	32
128	1.0	8	8	8	8	8	8	8
128	1.5	8	8	8	16	16	16	16
128	19.8	32	32	32	32	32	32	32
256	1.0	8	8	8	16	16	16	16
256	1.5	8	16	16	16	16	16	16
256	19.8	32	32	64	64	64	64	64
1024	1.0	16	16	16	32	32	32	32
1024	1.5	16	16	32	32	32	32	32
1024	19.8	64	64	64	128	128	128	128

$(\gamma_2/\gamma_1) = \tilde{\alpha}$, we have

$$(18)\ \tilde{T}^A_{IAP}(m,n,\gamma_1,\gamma_2) = 4\tau \sum_{k=1}^{m/\gamma_2} \sum_{j=1}^{\tilde{\alpha}} \sum_{i=1}^{\gamma_1} \left(\frac{m}{\gamma_1} - \tilde{\alpha}(k-1) - j\right)\left(\frac{n}{\gamma_2} - k + 1\right)$$
$$= \frac{2m^2(3n-m)}{3p}\tau + \frac{m^2}{\gamma_1}\tau + \frac{m^2 - 2mn}{\gamma_2}\tau + O(m)\,,$$

$$(19)\quad \tilde{T}^C_{IAP}(m,n,\gamma_1,\gamma_2) = \beta\sum_{k=1}^{m} 2d_2 + \lambda \sum_{k=1}^{m/\gamma_1} \sum_{j=1}^{\gamma_1} 2d_2 \left(\frac{m}{\gamma_1} - k + 1\right)$$
$$= 2md_2\beta + \left(\frac{m^2}{\gamma_1} + m\right) d_2\lambda\,,$$

$$(20)\quad \tilde{T}^A_{CMP}(m,n,\gamma_1,\gamma_2) = 2\tau \sum_{k=1}^{m/\gamma_2} \sum_{j=1}^{\gamma_2} d_1 \left(\frac{n}{\gamma_2} - k + 2\right)$$
$$= \left(\frac{2mn - m^2}{\gamma_2} + 3m\right) d_1\tau\,,$$

and

$$(21)\quad \tilde{T}^C_{CMP}(m,n,\gamma_1,\gamma_2) = \beta\sum_{k=1}^{m} 2d_1 + \lambda \sum_{k=1}^{m/\gamma_2} \sum_{j=1}^{\gamma_2} 2d_1 \left(\frac{n}{\gamma_2} - k + 2\right)$$
$$= 2md_1\beta + \left(\frac{2mn - m^2}{\gamma_2} + 3m\right) d_1\lambda\,.$$

The total parallel arithmetic cost and communication cost are thus given by

$$(22)\quad \begin{aligned} \tilde{T}_{II}^{A}(m,n,\gamma_1,\gamma_2) &= \tilde{T}_{IAP}^{A}(m,n,\gamma_1,\gamma_2) + \tilde{T}_{CMP}^{A}(m,n,\gamma_1,\gamma_2) \\ &= \frac{2m^2(3n-m)}{3p}\tau + \left(\frac{2mn-m^2}{\gamma_2}\right) d_1\tau + \frac{m^2}{\gamma_1}\tau \\ &\quad + \left(\frac{m^2-2mn}{\gamma_2}\right)\tau + O(md_1) \end{aligned}$$

and

$$(23)\quad \begin{aligned} \tilde{T}_{II}^{C}(m,n,\gamma_1,\gamma_2) &= \tilde{T}_{IAP}^{C}(m,n,\gamma_1,\gamma_2) + \tilde{T}_{CMP}^{C}(m,n,\gamma_1,\gamma_2) \\ &= 2nd\beta + \left(\frac{2mn-m^2}{\gamma_2}\right) d_1\lambda + \frac{m^2}{\gamma_1} d_2\lambda \\ &\quad + O(md)\,. \end{aligned}$$

When $m \le n$, the parallel time of Algorithm II can again be expressed as a function of m, n, γ_1, and p given as follows:

$$(24)\quad \begin{aligned} \tilde{T}_{II}(m,n,\gamma_1,p) &= \tilde{T}_{II}^{A}(m,n,\gamma_1,\gamma_2) + \tilde{T}_{II}^{C}(m,n,\gamma_1,\gamma_2) \\ &= \frac{2m^2(3n-m)}{3p}\tau + \left(\frac{2mn-m^2}{\gamma_2}\right) d_1(\tau+\lambda) \\ &\quad + \frac{m^2}{\gamma_1}(\tau + d_2\lambda) + \left(\frac{m^2-2mn}{\gamma_2}\right)\tau + O(m\log_2 p) \\ &= \frac{2m^2(3n-m)}{3p}\tau + \left(\frac{2mn-m^2}{p}\right) d_1\gamma_1(\tau+\lambda) \\ &\quad + \frac{m^2}{\gamma_1}(\tau + (d-d_1)\lambda) + \left(\frac{m^2-2mn}{p}\right)\gamma_1\tau \\ &\quad + O(md)\,. \end{aligned}$$

The value of γ_1 that minimizes $\tilde{T}_{II}(m,n,\gamma_1,p)$ in (24) can now be found by setting

$$\frac{\partial \tilde{T}_{II}(m,n,\gamma_1,p)}{\partial \gamma_1} = 0\,.$$

By noting that $d = \log_2 p$, $d_1 = \log_2 \gamma_1$, and $\partial d_1/\partial\gamma_1 = (\ln\gamma_1)/a$, where $a = \ln 2$, we obtain

$$(25)\quad \begin{aligned} \tilde{f}\left(p,\gamma_1,\frac{n}{m},\frac{\tau}{\lambda}\right) &= a\left(\frac{\tau}{\lambda}+1\right)\left(2\frac{n}{m}-1\right)(\ln\gamma_1+1){\gamma_1}^2 \\ &\quad - \frac{\tau}{\lambda}\left(2\frac{n}{m}-1\right){\gamma_1}^2 + ap(\ln\gamma_1) - p\left(\frac{\tau}{\lambda} + \log_2 p + a\right) \\ &= 0\,. \end{aligned}$$

Similarly, for fixed p, n/m, and τ/λ, the value of γ_1 that minimizes $\tilde{T}_{II}(m,n,\gamma_1,p)$ can be obtained by finding the solution to $\tilde{f}\,(p,\gamma_1,n/m,\tau/\lambda) = 0$. As before, γ_1 must be an integral power of 2, and we choose it as close to the solution of (25) as possible.

In order to see how the optimal γ_1 varies with the relative speeds of computation and communication, in Table 3 we list the numerical solution to $\tilde{f}\,(p,\gamma_1,n/m,\tau/\lambda) = 0$ for different values of τ/λ when p and n/m remain fixed. The optimal γ_1's chosen based on these numerical solutions are displayed in Table 4. Again note that the optimal $d_1 = \log_2\gamma_1$ is quite insensitive to the ratios of τ to λ.

TABLE 3
Numerical solution to $\tilde{f}(p, \gamma_1, n/m, \tau/\lambda) = 0$.

$m \leq n$		Different values of τ/λ						
p	n/m	1000	10	5	1	0.2	0.1	0
16	1.0	2.86	3.06	3.20	3.62	3.89	3.94	4.0
16	1.5	2.24	2.40	2.51	2.84	3.05	3.09	3.13
16	19.8	0.98	1.02	1.04	1.10	1.14	1.14	1.15
64	1.0	4.86	5.42	5.80	6.95	7.70	7.84	8.0
64	1.5	3.71	4.16	4.46	5.35	5.93	6.04	6.16
64	19.8	1.37	1.51	1.59	1.85	2.01	2.03	2.06
128	1.0	6.42	7.30	7.89	9.68	10.85	11.07	11.31
128	1.5	4.86	5.56	6.02	7.40	8.30	8.47	8.65
128	19.8	1.68	1.90	2.04	2.46	2.71	2.75	2.81
256	1.0	8.53	9.89	10.78	13.50	15.29	15.62	16.0
256	1.5	6.42	7.48	8.18	10.26	11.63	11.88	12.17
256	19.8	2.10	2.45	2.67	3.30	3.69	3.76	3.84
1024	1.0	15.34	18.41	20.40	26.41	30.40	31.15	32.0
1024	1.5	11.42	13.79	15.32	19.90	22.91	23.47	24.11
1024	19.8	3.46	4.23	4.70	6.07	6.93	7.09	7.27

TABLE 4
Predicted optimal γ_1 *when* $m \leq n$.

$m \leq n$		Different values of τ/λ						
p	n/m	1000	10	5	1	0.2	0.1	0
16	1.0	2	4	4	4	4	4	4
16	1.5	2	2	2	2	4	4	4
16	19.8	1	1	1	1	1	1	1
64	1.0	4	4	4	8	8	8	8
64	1.5	4	4	4	4	4	8	8
64	19.8	1	2	2	2	2	2	2
128	1.0	8	8	8	8	8	8	8
128	1.5	8	8	8	8	8	8	8
128	19.8	2	2	2	2	2	2	2
256	1.0	8	8	8	16	16	16	16
256	1.5	8	8	8	8	8	8	16
256	19.8	2	2	2	4	4	4	4
1024	1.0	16	16	16	32	32	32	32
1024	1.5	8	16	16	16	16	16	32
1024	19.8	4	4	4	8	8	8	8

Comparing the leading term of $T_{II}(m, n, \gamma_1, p)$ with the leading term of $T_s(m, n)/p$ for the case $m \geq n$, and from comparing the leading term of $\tilde{T}_{II}(m, n, \gamma_1, p)$ with the leading term of $\tilde{T}_s(m, n)/p$ for the case $m \leq n$, it can be concluded that Algorithm II is optimal in its leading term.

6.3. Analysis of storage requirements. According to our data mapping strategy for Algorithm II, the rows and columns of a given matrix are wrap mapped to γ_1 and γ_2 processors, respectively. Therefore, the processors will run out of rows and/or columns one by one in the last γ_1 or γ_2 elimination stages. As explained earlier, our communication algorithm requires all processors to participate in maintaining data proximity in every stage. We thus adopted the strategy of assigning one more row of all zeros and one more column of all zeros to each processor. The largest submatrix assigned to a processor is therefore $\lceil (m/\gamma_1) + 1 \rceil$ by $\lceil (n/\gamma_2) + 1 \rceil$. In addition to storing the submatrix, each processor also needs buffer space for sending and receiving the multipliers (in the IAP) and sending, receiving, and saving the pivot row (in the CMP). There is also integer overhead incurred by choosing particular data structures that facilitate a clean implementation. Such overhead amounts to $(2m/\gamma_1 + n/\gamma_2 + O(p))$ more integers in our implementation. In the analysis below we consider the total storage requirement on a single processor as the sum of the primary storage for data and the overhead storage for buffers, the extra zero row and column, and the integer overhead. The low-order terms that neither vary with m nor vary with n are ignored. The total storage is thus a function of m, n, γ_1, and γ_2. Since $\gamma_2 = p/\gamma_1$, for a given $m \times n$ matrix it is desirable to find the value of γ_1 that minimizes the total storage. We assume that the space for storing an integer is the same as the space for storing a floating-point number in the following analysis.

LEMMA 6.1. *For any given m, n, and $p = \gamma_1 \times \gamma_2$, the total storage requirement of Algorithm* II *on each node processor is given by*

$$\frac{mn}{p} + 7\frac{m}{\gamma_1} + 4\frac{n}{\gamma_2} + 7 \quad \text{if } 2\left(\frac{m}{\gamma_1} + 1\right) \geq \frac{n}{\gamma_2} + 1\,,$$

and

$$\frac{mn}{p} + 3\frac{m}{\gamma_1} + 6\frac{n}{\gamma_2} + 5 \quad \text{if } 2\left(\frac{m}{\gamma_1} + 1\right) \leq \frac{n}{\gamma_2} + 1\,.$$

Proof. As noted earlier, each processor is assigned a submatrix of size $(m/\gamma_1 + 1) \times (n/\gamma_2 + 1)$. The buffer space for sending and receiving the multipliers is twice the size of the largest set of multipliers, i.e., $2 \times 2\,(m/\gamma_1 + 1)$. Similarly, the buffer space for sending and receiving the pivot row is twice the row length of each submatrix, i.e., $2 \times (n/\gamma_2 + 1)$. Since the buffer space for multipliers can be reused for sending and receiving pivot rows, it is sufficient to have enough storage for the larger one of these two buffers. In addition to the buffer space for sending and receiving the pivot row, in the CMP we need extra buffer space of $2 \times (n/\gamma_2 + 1)$ floating-point numbers to save the pair of rows in case the updating is delayed. Summing up the data storage, buffer storage, and the integer overhead given above we obtain the results in the lemma. $\square$

THEOREM 6.2. *For any given m, n and p, the storage requirement of Algorithm* II *is minimized by $\gamma_1 = 2^k$, where $k \in [0, \log_2 p]$ is chosen so that γ_1 is as close as possible to $\sqrt{7mp/(6n)}$.*

Proof. Assuming that the buffer space for the multipliers and the pivot row cannot be overlapped, we seek γ_1 to minimize

$$S(m,n,\gamma_1,\gamma_2) = \frac{mn}{p} + 7\frac{m}{\gamma_1} + 6\frac{n}{\gamma_2} + 7\,.$$

Substituting $\gamma_2 = p/\gamma_1$, and setting

$$\frac{\partial S(m,n,\gamma_1,\gamma_2)}{\partial \gamma_1} = 0\,,$$

we obtain

$$\gamma_1 = \sqrt{\frac{7mp}{6n}}\,. \qquad \square$$

Recall that the data of the coefficient matrix only require storage for mn/p floating-point numbers per processor. Thus it is necessary to address the question of whether the overhead storage is a significant fraction of the primary storage for the chosen γ_1. In Corollary 6.3, we give the formula for computing the ratio of the overhead storage to the primary storage mn/p when $\gamma_1 = \sqrt{7mp/(6n)}$.

COROLLARY 6.3. *When $\gamma_1 = \sqrt{7mp/(6n)}$, the ratio of the overhead storage to the primary storage is given by*

$$10.8\sqrt{\frac{p}{mn}} + 7\frac{p}{mn} \quad \textit{if } 2\left(\frac{m}{\gamma_1}+1\right) \geq \frac{n}{\gamma_2}+1\,,$$

and

$$9.3\sqrt{\frac{p}{mn}} + 5\frac{p}{mn} \quad \textit{if } 2\left(\frac{m}{\gamma_1}+1\right) \leq \frac{n}{\gamma_2}+1\,.$$

Proof. Substituting γ_1 in Lemma 6.1 by $\sqrt{7mp/(6n)}$ and γ_2 by p/γ_1, we obtain

$$S(m,n,\gamma_1,p) = \frac{mn}{p} + 10.8\sqrt{\frac{mn}{p}} + 7\,,$$

and

$$\tilde{S}(m,n,\gamma_1,p) = \frac{mn}{p} + 9.3\sqrt{\frac{mn}{p}} + 5\,.$$

The results in the corollary are obtained by computing

$$\frac{S(m,n,\gamma_1,p) - (mn/p)}{mn/p}$$

and

$$\frac{\tilde{S}(m,n,\gamma_1,p) - (mn/p)}{mn/p}\,. \qquad \square$$

Since γ_1 is unlikely to be exactly equal to $\sqrt{7mp/(6n)}$ in practice, we computed the actual overhead storage and compared with the results obtained from the formula given by Corollary 6.3. Letting γ_1^s denote the γ_1 chosen by Theorem 6.2, in Table 5 we list the values of $\sqrt{7mp/(6n)}$, γ_1^s and the predicted and actual ratio of overhead storage to primary storage for a set of matrices. The two ratios given as the predicted percentages are obtained by substituting $\gamma_1 = \sqrt{7mp/(6n)}$ in each of the two formulas given in Corollary 6.3. The actual percentage given in the last column is computed by substituting the chosen γ_1^s into the appropriate formula in Lemma 6.1.

TABLE 5
Predicted and actual overhead storage.

p	m	n	$\sqrt{7mp/(6n)}$	Predicted percentage	γ_1^s	Actual percentage
16	500	500	4.32	7.5%–8.7%	4	8.8%
16	600	400	5.29	7.6%–8.9%	4	9.7%
16	400	600	3.53	7.6%–8.9%	4	8.7%
64	1000	1000	8.64	7.5%–8.7%	8	8.8%
64	1200	800	10.57	7.6%–8.9%	8	9.7%
64	800	1200	7.05	7.6%–8.9%	8	8.7%
64	1980	100	38.44	16.9%–19.6%	32	20.7%
64	100	1980	1.93	16.9%–19.6%	2	19.5%

TABLE 6
Predicted γ_1^s and predicted optimal γ_1.

				Different values of τ/λ						
				1000	10	5	1	0.2	0.1	0
p	m	n	γ_1^s	γ_1	γ_1	γ_1	γ_1	γ_1	γ_1	γ_1
16	500	500	4	2	4	4	4	4	4	4
16	600	400	4	4	4	4	4	4	4	4
16	400	600	4	2	2	2	2	4	4	4
64	1000	1000	8	4	4	4	8	8	8	8
64	1200	800	8	8	8	8	8	8	8	8
64	800	1200	8	4	4	4	4	4	8	8
64	1980	100	32	16	16	32	32	32	32	32
64	100	1980	2	1	2	2	2	2	2	2

For easy comparison, in Table 6 we give the value of γ_1^s as well as the predicted optimal γ_1 for the set of matrices listed in Table 5.

Corollary 6.3 implies that the overhead storage will be insignificant if m, n are large and $p \ll \min\{m, n\}$.

7. Numerical experiments. Our experiments were performed on a 64-processor Intel iPSC Hypercube, and Algorithm II was implemented in FORTRAN. Note that Algorithm I is a special case of Algorithm II when the subcube-grid has dimension p by 1. The programs were compiled using the Ryan–McFarland FORTRAN compiler. We provide timing results for single-precision and double-precision implementations. The maximum run time over all the node processors is reported as the parallel execution time for each test problem. Note that the factorization time reported does not include the time for initialization and data generation.

The execution times (in seconds) of the serial and parallel algorithms are denoted by T_s and T, respectively, and as in previous sections, m and n denote the number of rows and columns of each test matrix and γ_1 and γ_2 denote the number of processors along each dimension of the two-dimensional grid embedded in the hypercube.

Our experiments were designed to measure speedup, and demonstrate how the aspect ratio of the subcube-grid affects the performance of Algorithm II. We show that when the predicted optimal aspect ratios are used, the execution time and the storage requirement either coincide with or are very close to the actual minimum as the theory developed in previous sections predicts.

7.1. The measurement of serial time. Table 7 reports the serial time T_s for each randomly generated test matrix. Note that when $\gamma_1 = \gamma_2 = 1$, the two-dimensional processor grid degenerates to a single processor, and Algorithm II involves only the Independent Annihilation Phase (IAP) on a single processor. Since inter-processor communication is not needed during the IAP, the code for the IAP running on one node indeed implements the sequential Givens algorithm. We thus measure T_s by the execution time of the parallel code running on a 1×1 grid.

TABLE 7
Execution times of the sequential Givens algorithm.

Single precision			Double precision		
m	n	T_s (sec)	m	n	T_s (sec)
100	100	67.500	50	50	10.800
125	125	130.465	75	75	35.400
150	150	223.890	100	100	82.600
175	175	353.760	125	125	160.000
200	200	526.095	150	150	274.800
90	60	26.830	60	40	10.100
120	80	62.010	90	60	32.640
135	90	87.550	120	80	75.700
160	120	174.600	135	90	107.020
240	160	477.510	160	120	213.905
60	90	25.300	40	60	9.360
80	120	59.300	60	90	30.995
90	135	84.200	80	120	72.800
120	160	170.600	90	135	103.310
160	240	467.000	120	160	209.510

Due to the limited memory of 512 kbytes on a single node, the largest matrix we could factor using one processor was about 200×200 in single precision or 150×150 in double precision. In order to measure the speedup and efficiency of the parallel algorithm, we need to estimate the serial factorization time of much larger matrices. For any square matrix of dimension n, we approximate the factorization time using the formulae

$$T_s(n) \approx c_1 n^3 + c_2 n^2 + c_3 n + c_4 \,, \tag{26}$$

where c_1, c_2, c_3, and c_4 are obtained in the following manner. First note that by equating $T_s(n)$ to the known execution times for $n = 100, 125, 150, 175$, and 200 (for single-precision implementation) or $n = 50, 75, 100, 125$, and 150 (for double-precision implementation), we obtain five equations and four unknowns. By finding the least-squares solution to the overdetermined system of equations we obtain the coefficients $\{c_1, c_2, c_3, c_4\}$. The estimated $T_s(n)$ are compared with the actual execution times in Table 8. Since the node processors on the hypercube do not support multiprogramming, the execution times measured on a node are consistent and reproducible. This feature allows us to obtain accurate estimates based on a relatively small set of samples.

TABLE 8
Measured and estimated times of the sequential Givens algorithm.

Single precision				Double precision			
m	n	T_s (sec)	Estimated T_s	m	n	T_s (sec)	Estimated T_s
100	100	67.500	67.500 sec	50	50	10.800	10.806 sec
125	125	130.465	130.467 sec	75	75	35.400	35.377 sec
150	150	223.890	223.887 sec	100	100	82.600	82.634 sec
175	175	353.760	353.762 sec	125	125	160.000	159.977 sec
200	200	526.095	526.095 sec	150	150	274.800	274.806 sec

7.2. The effect of the aspect ratio of the subcube-grid. In this section we present numerical experiments to demonstrate the effect on the execution time of Algorithm II induced by varying the aspect ratio of the processor grid. Table 9 gives the timing results obtained from the single-precision implementation of Algorithm II. Table 10 gives the double-precision timing results. The minimum execution time for each test matrix is marked by an asterisk (*).

Recall that for given m, n, and p, the predicted optimal γ_1 varies for different values of τ/λ. In Table 2 and Table 4 we computed the predicted values of the optimal γ_1 for the ratios of τ/λ ranging from 0 ($\tau \ll \lambda$) to 1000 ($\tau \gg \lambda$). For easy comparison with the actual optimal execution time T^*, we let γ_1^ℓ denote the smallest predicted optimal γ_1 and γ_1^u denote the largest predicted optimal γ_1, and label the execution time corresponding to γ_1^ℓ or γ_1^u as T_ℓ or T_u, respectively, for each test matrix in Tables 9 and 10.

Some timing results are missing in the tables. In some cases, we did not obtain the execution time because of storage limitation. In particular, the maximum number of bytes that may be sent in a single message on the hypercube is 16K bytes (4000 single-precision or 2000 double-precision floating-point numbers) and this limit was exceeded for some choices of γ_1 and γ_2. In other cases, for the very large test problems whose factorization is very expensive, we only provide the timing result for the optimal choice of $\gamma_1 \times \gamma_2$ because the effect of the shape of the grid on speedup and efficiency has been well demonstrated on smaller problems.

Since it may be equally important to minimize the storage requirement on each node processor, it is desirable that γ_1^s in Theorem 6.2 coincides with the choice of γ_1, which minimizes the execution time. In order to see how Algorithm II performs in this aspect, we label the execution time corresponding to γ_1^s as $T_\dagger$ for each test matrix in Tables 9 and 10.

It is interesting to see that $T_\dagger$, T_ℓ, or T_u either coincide with or are very close to the actual optimal T^* for all test matrices in Tables 9 and 10. It is also worth noting that by embedding an appropriate processor grid we not only minimize the storage usage and communication/computation cost of the parallel algorithm, but also help balance the work load and reduce processor idle time. The 1980×100 and 100×1980 matrices are examples to demonstrate how a proper choice of γ_1 can reduce the processor idle time. Clearly the choice of a 1×64 grid for the 1980×100 matrix is equivalent to wrapping the 100 columns around the 64 processors where each processor is assigned one column or two columns. In contrast, the choice of a 64×1 grid for the same matrix will assign 30 or 31 rows to each processor. In the former case, because only 36 processors are assigned two columns, starting from the 37th elimination stage, the 64 processors will become idle one by one after each following elimination stage. In the latter case, since only one row from the 30 rows or two rows from the 31 rows could be the pivot rows, each processor has 29 to 31 rows of data to process at each of the 100 elimination stages. The reduction of idle time is thus quite significant while using a 64×1 grid for this example. A similar argument applies to the 100×1980 example.

TABLE 9
Single-precision execution times of Algorithm II.

Single-precision execution times (sec), $\gamma_1 \times \gamma_2 = 64$								
m	n	64×1	32×2	16×4	8×8	4×16	2×32	1×64
1000	1000	1493	1250	1155	$1146^{*}_{\dagger,u}$	1185_{ℓ}	1304	1579
1700	1700				$5258^{*}_{\dagger,u}$			
1200	800	1224	1075	1021*	$1041_{\dagger,u,\ell}$	1103	1264	1628
800	1200	1493	1178	1051	$1007^{*}_{\dagger,u}$	1028_{ℓ}	1096	1270
1980	100	43.8*	$44.8_{\dagger,u}$	48.7_{ℓ}	60.3	85.4	138.9	251.6
100	1980	196.9	102.4	62.7	46.7	41	$39.4^{*}_{\dagger,u}$	40.1_{ℓ}

*The minimum execution time.

$_{\ell}$Execution time when employing the smallest predicted optimal γ_1.

$_{u}$Execution time when employing the largest predicted optimal γ_1.

$_{\dagger}$Execution time when employing γ_1^s to minimize storage.

TABLE 10
Double-precision execution times of Algorithm II.

Double-precision execution times (sec), $\gamma_1 \times \gamma_2 = 16$						
m	n	16×1	8×2	4×4	2×8	1×16
500	500	729	680	$676.5^{*}_{\dagger,u}$	702_{ℓ}	779
600	400	631	605*	$615_{\dagger,u,\ell}$	656	757
400	600	686	617	$599^{*}_{\dagger,u}$	610_{ℓ}	657

Double-precision execution times (sec), $\gamma_1 \times \gamma_2 = 64$								
m	n	64×1	32×2	16×4	8×8	4×16	2×32	1×64
1000	1000	1858	1549	1423	$1402^{*}_{\dagger,u}$	1448_{ℓ}	1586	1918
1200	1200				$2357^{*}_{\dagger,u}$			
1200	800	1517	1326	1256*	$1270_{\dagger,u,\ell}$			
800	1200	1861		1302	$1238^{*}_{\dagger,u}$	1259_{ℓ}		
1980	100	52.1*	$52.6_{\dagger,u}$	57.6_{ℓ}	71.0	101.5	164.4	
100	1980	249.6	129.8	78.4	57.9	49.9	$47.0^{*}_{\dagger,u}$	48.1_{ℓ}

*The minimum execution time.

$_{\ell}$Execution time when employing the smallest predicted optimal γ_1.

$_{u}$Execution time when employing the largest predicted optimal γ_1.

$_{\dagger}$Execution time when employing γ_1^s to minimize storage.

TABLE 11
Estimated speedup and efficiency of Algorithm II.

Single precision							
$p = 64$, $m = n = 1000$ $T_s \approx 64,377$ sec = 17 hr 52 min 57 sec							
$\gamma_1 \times \gamma_2$	64×1	32×2	16×4	8×8	4×16	2×32	1×64
T (sec) speedup efficiency	1493 43.1 67%	1250 51.5 81%	1155 55.7 87%	1146* 56.2* 88%*	1185 54.3 85%	1304 49.4 77%	1579 40.8 64%
$p = 64$, $m = n = 1700$ $T_s \approx 315,578$ sec = 3 days 15 hr 39 min 38 sec							
$\gamma_1 \times \gamma_2$	64×1	32×2	16×4	8×8	4×16	2×32	1×64
T (sec) speedup efficiency				5258* 60.0* 94%*			

*The minimum execution time.

Tables 11 and 12 report the estimated speedup and efficiency for a set of test matrices. The speedup and efficiency are each computed using

$$\text{speedup} = \frac{\text{Estimated } T_s}{T},$$

and

$$\text{efficiency} = \frac{\text{speedup}}{p},$$

where p is the total number of processors employed, and $p = \gamma_1 \times \gamma_2$.

7.3. Further enhancement. In [16] Pothen and Raghavan proposed a hybrid algorithm for performing orthogonal decomposition of a rectangular matrix on local-memory multiprocessors. The hybrid algorithm proposed in [16] can be viewed as a variant of Algorithm I. The difference lies in the following two aspects. In the IAP the hybrid scheme uses Householder transformations instead of Givens rotations to reduce the arithmetic cost. In the CMP the hybrid scheme used a different communication scheme in merging the local pivot rows. Since Algorithm I is a special case of Algorithm II when a p-by-1 grid is embedded in the hypercube, the strategy of applying Householder transformations in the IAP can be used to reduce the arithmetic cost of Algorithm II regardless of the choice of γ_1. Furthermore, when $\gamma_2 > 1$, the use of Householder transformations during the IAP can also reduce the communication cost of Algorithm II because there are only half as many multipliers to be communicated within each subcube. In terms of message length, each message to be sent and received during the IAP is reduced by a factor of 2 when Householder transformations are used.

As far as our implementation of Algorithm II is concerned, the code for the IAP involves one single subroutine implementing Givens rotations. Therefore, an enhanced version of Algorithm II is immediately obtained by recoding this subroutine using Householder transformations. Note that our communication algorithms and the entire CMP of Algorithm II remain unchanged. In this section we report timing results of the enhanced Algorithm II and compare its performance with other schemes.

TABLE 12
Estimated speedup and efficiency of Algorithm II.

Double precision							
$p = 16$, $m = n = 500$, $T_s \approx 9,962$ sec = 2 hr 46 min 2 sec							
$\gamma_1 \times \gamma_2$	16×1		8×2	4×4	2×8	1×16	
T (sec)	729		680	676.5*	702	779	
speedup	13.7		14.7	14.7*	14.2	12.8	
efficiency	86%		92%	92%*	89%	80%	
$p = 64$, $m = n = 1000$, $T_s \approx 79,318$ sec = 22 hr 1 min 58 sec							
$\gamma_1 \times \gamma_2$	64×1	32×2	16×4	8×8	4×16	2×32	1×64
T (sec)	1858	1549	1423	1402*	1448	1586	1918
speedup	42.7	51.2	55.7	56.6*	54.8	50.0	41.3
efficiency	67%	80%	87%	89%*	86%	78%	65%
$p = 64$, $m = n = 1200$, $T_s \approx 136,953$ sec = 1 day 14 hr 2 min 33 sec							
$\gamma_1 \times \gamma_2$	64×1	32×2	16×4	8×8	4×16	2×32	1×64
T (sec)				2357*			
speedup				58.1*			
efficiency				91%*			

*The minimum execution time.

TABLE 13
Execution times of the sequential Householder algorithm.

Single precision			Double precision		
m	n	T_s (sec)	m	n	T_s (sec)
100	100	60.1	50	50	9.2
125	125	115.3	75	75	29.3
150	150	196.9	100	100	67.5
175	175	310.1	125	125	129.6
200	200	460.0	150	150	221.4
90	60	23.9	60	40	8.5
120	80	54.7	90	60	26.8
135	90	77.0	120	80	61.4
160	120	153,0	135	90	86.5
240	160	415.6	160	120	172.0
60	90	23.2	40	60	8.1
80	120	53.5	60	90	26.1
90	135	75.5	80	120	60.1
120	160	151.3	90	135	84.8
160	240	411.0	120	160	170.1

When $\gamma_1 = \gamma_2 = 1$, the enhanced Algorithm II involves only the IAP phase on one node and thus implements the sequential Householder algorithm. The serial time T_s based on the Householder algorithm is therefore measured by the execution time of the parallel code running on a 1×1 grid.

Table 13 reports the execution times T_s of the sequential Householder algorithm for some randomly generated test matrices. We again estimated the serial factorization time T_s for large n-by-n matrices by choosing the coefficients for a cubic polynomial $T_s(n)$ as explained earlier in this section. We compare the estimated $T_s(n)$ with the actual execution times in Table 14.

In Tables 15 and 16 we show that the aspect ratio of the processor grid has a similar effect on the enhanced Algorithm II. An analysis similar to the one in §6 can be done in order to obtain reliable estimates for the best γ_1 to use in conjunction with the enhanced version of Algorithm II. Tables 17 and 18 report the "estimated" speedup and efficiency for a set of test matrices.

Next we compare the performance of Algorithm II and the enhanced Algorithm II in Tables 19 and 20. Note that Algorithm I is the special case of Algorithm II when the processor grid is chosen to be $p \times 1$. Therefore, the enhanced version of Algorithm I is a FORTRAN implementation (with a different communication scheme) of the hybrid algorithm proposed in [16]. In [16] Pothen and Raghavan implemented the hybrid algorithm in the *C* language and compared its performance with four other schemes including one based on the greedy Givens sequence. The latter can be viewed as a variant of Algorithm I with a different communication scheme.

TABLE 14
Measured and estimated times of the sequential Householder algorithm.

Single precision				Double precision			
m	n	T_s (sec)	Estimated T_s	m	n	T_s (sec)	Estimated T_s
100	100	60.1	60.1 sec	50	50	9.2	9.2 sec
125	125	115.3	115.3 sec	75	75	29.3	29.3 sec
150	150	196.9	196.9 sec	100	100	67.5	67.5 sec
175	175	310.1	310.1 sec	125	125	129.6	129.6 sec
200	200	460.0	460.0 sec	150	150	221.4	221.4 sec

The timing results listed in Table 19 indicate that the enhanced Algorithm II coupled with the optimal choice of γ_1 has the fastest execution time. The possible improvement in execution time by the hybrid scheme over Algorithm I can be seen by comparing the data in column 1 with the data in column 2. Note that when $m/p \ll n$ (e.g., $p = 64$, and $m \times n = 800 \times 1200$ or $m \times n = 100 \times 1980$), the hybrid scheme could become less efficient. The factor contributing to the longer execution time of the hybrid scheme is that in this case each submatrix to be reduced by Householder transformations has dimension $(m/p) \times n$ and when $(m/p) \ll n$, the saving by Householder transformations is relatively small and is less than the different overhead caused by employing Householder transformations instead of Givens rotations. This is not likely to happen when $\gamma_1 \times \gamma_2$ is chosen according to the shape of the matrix as demonstrated by the results for the enhanced Algorithm II shown in the same table. The possible improvement by the enhanced Algorithm II over Algorithm II can be seen by comparing the data in column 3 with the data in column 4. As noted earlier, when $\gamma_2 = 1$, the hybrid scheme has lower arithmetic cost but the same communication cost compared to Algorithm I; when $\gamma_2 > 1$, the enhanced Algorithm II not only has lower arithmetic cost but also has lower communication cost compared to Algorithm II. This observation is supported by the timing results in Table 19.

In Table 20 we list the storage requirement for each of the four schemes. The storage requirement of the enhanced Algorithm II is either the minimum or different

TABLE 15
Single-precision execution times of the enhanced Algorithm II.

The enhanced Algorithm II								
Single-precision execution times (sec), $\gamma_1 \times \gamma_2 = 64$								
m	n	64×1	32×2	16×4	8×8	4×16	2×32	1×64
1000	1000	1557	1215	1060	1011*	1017	1084	1257
1700	1700				4580*			
1200	800	1231	1012	922	905*	930	1030	1265
800	1200	1618	1183	986	907	891*	924	1030
1980	100	41.7	39.8*	42.0	48.5	64.4	99.6	175.0
100	1980	221.5	125.1	75.7	50.8	41.0	36.6	36.1*

*The minimum execution time.

TABLE 16
Double-precision execution times of the enhanced Algorithm II.

The enhanced Algorithm II						
Double-precision execution times (sec), $\gamma_1 \times \gamma_2 = 16$						
m	n	16×1	8×2	4×4	2×8	1×16
500	500	653	572	542*	551	596
600	400	549	500	488*	509	571
400	600	639	534	488	485*	508

Double-precision execution times (sec), $\gamma_1 \times \gamma_2 = 64$								
m	n	64×1	32×2	16×4	8×8	4×16	2×32	1×64
1000	1000	1819	1399	1214	1138*	1148	1224	1424
1200	1200				1905*			
1200	800	1429		1049	1021*	1051		
800	1200	1915			1031	1006*	1043	
1980	100	46.9	43.7*	45.7	54.5	72.0	113.4	
100	1980	274.5	152.8	89.3	60.6	45.8	40.8	40.4*

*The minimum execution time.

TABLE 17
Estimated speedup and efficiency of the enhanced Algorithm II.

The enhanced Algorithm II							
Single precision							
$p = 64$, $m = n = 1000$ $T_s \approx 55,464$ sec = 15 hr 24 min 24 sec							
$\gamma_1 \times \gamma_2$	64×1	32×2	16×4	8×8	4×16	2×32	1×64
T (sec)	1557	1215	1060	1011*	1017	1084	1257
speedup	35.6	45.7	52.3	54.9*	54.5	51.2	44.1
efficiency	56%	71%	82%	86%*	85%	80%	69%
$p = 64$, $m = n = 1700$ $T_s \approx 271,428$ sec = 3 days 3 hr 23 min 48 sec							
$\gamma_1 \times \gamma_2$	64×1	32×2	16×4	8×8	4×16	2×32	1×64
T (sec)				4580*			
speedup				59.3*			
efficiency				93%*			

*The minimum execution time.

TABLE 18
Estimated speedup and efficiency of the enhanced Algorithm II.

The enhanced Algorithm II							
Double precision							
$p = 16$, $m = n = 500$ $T_s \approx 7,873$ sec = 2 hr 11 min 13 sec							
$\gamma_1 \times \gamma_2$	16×1		8×2	4×4	2×8	1×16	
T (sec)	653		572	542*	551	596	
speedup	12.1		13.8	14.5*	14.3	13.2	
efficiency	76%		86%	91%*	89%	83%	
$p = 64$, $m = n = 1000$ $T_s \approx 62,426$ sec = 17 hr 20 min 26 sec							
$\gamma_1 \times \gamma_2$	64×1	32×2	16×4	8×8	4×16	2×32	1×64
T (sec)	1819	1399	1214	1138*	1148	1224	1424
speedup	34.3	44.6	51.4	54.9*	54.4	51.0	43.8
efficiency	54%	70%	80%	86%*	85%	80%	67%
$p = 64$, $m = n = 1200$ $T_s \approx 107,711$ sec = 1 day 5 hr 55 min 11 sec							
$\gamma_1 \times \gamma_2$	64×1	32×2	16×4	8×8	4×16	2×32	1×64
T (sec)				1905*			
speedup				56.5*			
efficiency				88%*			

*The minimum execution time.

from the minimum for less than 0.1 percent.

Finally, in view of the improvement in execution time and storage requirement by employing Householder transformations in the Independent Annihilation Phase of Algorithm II, the saving by reducing the length of each message in the IAP by a factor of 2 appears to be quite significant. Thus, instead of employing Householder transformations in the IAP, we might reduce the execution time and storage requirement of Algorithm II by simply storing the multiplier pair corresponding to each Givens rotation as a single real number using the economical storage technique proposed by Stewart in [21]. At the cost of compressing and retrieving the rotations, the parallel algorithm employing Givens rotations would have the same communication cost and storage requirement as the one employing Householder transformations in the IAP phase, and their performances would be comparable.

8.1. A summary. In this paper we considered the problem of factoring a dense rectangular matrix on a hypercube multiprocessor. The hypercube network is configured as a two-dimensional subcube-grid in the proposed algorithm. Our analysis of the algorithm determines how the aspect ratio of the subcube-grid should be chosen in order to minimize the execution time or storage usage. The algorithm was implemented in FORTRAN and tested on an Intel iPSC hypercube with 64 processors. Our numerical experiments demonstrate the effect of the aspect ratio on the performance of the parallel algorithm and show that the execution time or storage requirement using the predicted aspect ratio is very close to the actual minimum for

TABLE 19
Comparing the enhanced Algorithm II *with other schemes.*

Single-precision execution times (sec)						
p	m	n	$\gamma_1 \times \gamma_2 = p \times 1$		$\gamma_1 \times \gamma_2$ = optimal choice	
			Algorithm I	Hybrid	Algorithm II	Enhanced II
64	1000	1000	1493	1557	1146	1011*
64	1700	1700			5258	4580*
64	1200	800	1224	1231	1021	905*
64	1980	100	43.8	41.7	43.8	39.8*
64	800	1200	1493	1618	1007	891*
64	100	1980	196.9	221.5	39.4	36.1*
Double-precision execution times (sec)						
p	m	n	$\gamma_1 \times \gamma_2 = p \times 1$		$\gamma_1 \times \gamma_2$ = optimal choice	
			Algorithm I	Hybrid	Algorithm II	Enhanced II
16	500	500	729	653	676.5	542*
64	1000	1000	1858	1819	1402	1138*
64	1200	1200			2357	1905*
16	600	400	631	549	605	488*
64	1200	800	1517	1429	1256	1021*
64	1980	100	52.1	46.9	52.1	43.7*
16	400	600	686	639	599	485*
64	800	1200	1861	1915	1238	1006*
64	100	1980	249.6	274.5	47.0	40.4*

*The minimum execution time.

the test matrices.

Another feature of the algorithm proposed in this article is that redundant computations are incorporated in a communication scheme that takes full advantage of the hypercube topology. With the proposed communication scheme the data are always exchanged between neighboring processors. Furthermore, because the exchanges at each step involve distinct pairs of processors and employ separate communication channels, they can occur simultaneously. The latter feature is important in reducing traffic congestion in the network. It is expected that in future generations of hypercubes special hardware support may achieve a situation where sending a message to a processor several hops away may not take significantly longer than sending the message to a neighbor. However, the problem of traffic congestion will still exist. The communication scheme we proposed in this paper provides a solution to this problem.

The extensive experimental results presented in §7 also show that the proposed algorithm can be efficiently implemented and various enhancements can be easily incorporated to further reduce the execution time and storage requirement.

8.2. Further work. Recall that when we applied Algorithm II to a dense square matrix, substantial saving in execution time and storage usage were obtained by employing the hypercube as a subcube-grid with dimensions γ_1 being as close to γ_2 as possible. A natural question to ask is whether Algorithm II can be adapted to parallelize other numerical algorithms efficiently. In this section we give such an example by applying the ideas of Algorithm II to parallelize Gaussian elimination with pairwise pivoting on a hypercube multiprocessor. We briefly review the pairwise pivoting scheme and sketch how to adapt Algorithm II for this task.

TABLE 20
Comparing the enhanced Algorithm II *with other schemes.*

Storage requirement (in 4-byte words)						
Single-precision implementation						
p	m	n	$\gamma_1 \times \gamma_2 = p \times 1$		$\gamma_1 \times \gamma_2 =$ optimal choice	
			Algorithm I	Hybrid	Algorithm II	Enhanced II
64	1000	1000	22606	22606	17808	17546*
64	1700	1700	56750	56750	48258	47822*
64	1200	800	20618	20618	17246	16946*
64	1980	100	4366	4366	4366	4212*
64	800	1200	23394	23394	17108*	17121
64	100	1980	16390	16390	4313	4296*
Double-precision implementation						
p	m	n	$\gamma_1 \times \gamma_2 = p \times 1$		$\gamma_1 \times \gamma_2 =$ optimal choice	
			Algorithm I	Hybrid	Algorithm II	Enhanced II
16	500	500	38508	38508	34784	34260*
64	1000	1000	45212	45212	35616	35092*
64	1200	1200	61236	61236	50016	49392*
16	600	400	35756	35756	33660	33060*
64	1200	800	41236	41236	34492	33892*
64	1980	100	8732	8732	8732	8424*
16	400	600	38860	38860	33384*	33410
64	800	1200	46788	46788	34216*	34242
64	100	1980	32780	32780	8626	8592*

*The minimum execution time.

The method of Gaussian elimination using triangularization by elementary stabilized matrices constructed by pairwise pivoting is analyzed by Sorensen in [20]. It is shown that a variant of this scheme that is suitable for implementation on a parallel computer is numerically stable although the error bound is larger than the one for the standard partial pivoting algorithm. The serial algorithm and its analysis are given in detail in [20]. For our purpose, it is sufficient to note that the variant we are considering can be understood as applying a 2×2 elementary matrix to each pair of rows in a fashion similar to applying Givens rotations. Recall that in the Givens scheme, we apply the rotation of the following form to a pair of rows to annihilate a leading nonzero element from one of the rows:

$$S = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} .$$

For Gaussian elimination with pairwise pivoting, this elementary 2×2 matrix will be of the following form:

$$S = \begin{pmatrix} 1 & 0 \\ \gamma & 1 \end{pmatrix} P ,$$

where P is a 2×2 permutation. Therefore, to annihilate one element, only one row of data is modified. The serial arithmetic cost is therefore one-half of the Givens scheme. If the work load is evenly distributed among the multiple processors, then the parallel arithmetic cost is also one-half of the parallel Givens scheme. We can further improve the numerical stability without any cost by performing partial pivoting whenever parallelism can be maintained.

Following our description of Algorithm II in §5, we shall have each processor perform Gaussian elimination with "partial pivoting" in the IAP phase at each reduction step. After that all of the processors can cooperate to perform Gaussian elimination with "pairwise pivoting" in the CMP phase to eliminate the leading nonzeros in the local pivot rows. Note that with the wrap mapping a balanced work load distribution can be maintained throughout the entire elimination process as long as the kth row of A is reduced to the kth row of the upper triangular factor [4]. Therefore, explicit permutations during the CAP at the kth reduction step are needed only when the pair of rows involves row k and row k is not chosen as the pivot row. Whenever this happens, our communication scheme ensures that both rows are present in the two processors involved. The explicit permutation can thus be done at no extra cost by carefully delaying the actual modification until the very last step. Another point worthy of noting is that there is *no* redundant computation involved simply because the row to be further exchanged is not modified. The analysis of the parallel scheme would be similar to the performance analysis of Algorithm II.

REFERENCES

[1] H. M. AHMED, J.-M. DELOSME, AND M. MORF, *Highly concurrent computing structures for matrix arithmetic and signal processing*, Computer, 15 (1982), pp. 65–82.

[2] A. BOJANCZYK, R. P. BRENT, AND H. T. KUNG, *Numerically stable solution of dense systems of linear equations using mesh-connected processors*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 95–104.

[3] R. M. CHAMBERLAIN AND M. J. D. POWELL, QR *factorization for linear least squares problems on the hypercube*, Tech. Report. CCS 86/10, Department of Science and Technology, Chr. Michelsen Institute, Bergen, Norway, 1986.

[4] 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.

[5] ———, QR *factorization of a dense matrix on a shared-memory multiprocessor*, Parallel Comput., 11 (1989), pp. 55–71.

[6] M. COSNARD, J.-M. MULLER, AND Y. ROBERT, *Parallel* QR *decomposition of a rectangular matrix*, Numer. Math., 48 (1986), pp. 239–249.

[7] G. J. DAVIS, *Column* LU *factorization with pivoting on a hypercube multiprocessor*, Tech. Report 6219, Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, TN, 1985.

[8] J. J. DONGARRA, A. H. SAMEH, AND D. C. SORENSEN, *Implementation of some concurrent algorithms for matrix factorization*, Parallel Comput., 3 (1985), pp. 25–34.

[9] L. ELDEN, *A parallel* QR *decomposition algorithm*, Tech. report, Department of Scientific Computing, Uppsala University, and Department of Mathematics, Linkoping University, October 1987.

[10] G. A. GEIST AND M. T. HEATH, *Parallel Cholesky factorization on a hypercube multiprocessor*, Tech. Report ORNL-6211, Oak Ridge National Laboratory, Oak Ridge, TN, 1985.

[11] W. M. GENTLEMAN AND H. T. KUNG, *Matrix triangularization by systolic arrays*, in Real Time Signal Processing IV: SPIE Proceeding, Vol. 298, Bellingham, WA, 1981, Society of Photo-Optical Instrumentation Engineers, pp. 19–26.

[12] M. T. HEATH, *Parallel Cholesky factorization in message passing multiprocessor environments*, Tech. Report ORNL-6150, Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, TN, 1985.

[13] R. E. LORD, J. S. KOWALIK, AND S. P. KUMAR, *Solving linear algebraic equations on an* MIMD *computer*, J. Assoc. Comput. Mach., 30 (1983), pp. 103–117.

[14] F. T. LUK, *A rotation method for computing the* QR-*decomposition*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 452–459.

[15] J. J. MODI AND M. R. B. CLARKE, *An alternative Givens ordering*, Numer. Math., 43 (1984), pp. 83–90.

[16] A. POTHEN AND P. RAGHAVAN, *Distributed orthogonal factorization: Givens and Householder algorithms*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 1113-1134.

[17] A. POTHEN, J. SOMESH, AND U. VEMULAPATI, *Orthogonal factorization on a distributed memory multiprocessor*, in Proc. Hypercube Multiprocessors 1987, M. T. Heath, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987, pp. 587–596.

[18] A. H. SAMEH AND D. J. KUCK, *A parallel* QR *algorithm for symmetric tridiagonal matrices*, IEEE Trans. Comput., C-26 (1977), pp. 147–153.

[19] ———, *On stable parallel linear system solvers*, J. Assoc. Comput. Mach., 25 (1978), pp. 81–91.

[20] D. C. SORENSEN, *Analysis of pairwise pivoting in Gaussian elimination*, Tech. Report ANL/MCS-TM-26, Argonne National Laboratory, Argonne, IL, February 1984.

[21] G. W. STEWART, *The economical storage of plane rotations*, Numer. Math., 25 (1976), pp. 137–138.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 6, pp. 1029–1052, November 1990

DOMAIN DECOMPOSITION METHODS FOR SYSTEMS OF CONSERVATION LAWS: SPECTRAL COLLOCATION APPROXIMATIONS*

ALFIO QUARTERONI†

Abstract. Hyperbolic systems of conservation laws that are discretized in space by spectral collocation methods and advanced in time by finite difference schemes are considered. At any time-level a domain decomposition method is introduced that is based on an iteration-by-subdomain procedure yielding at each step a sequence of independent subproblems (one for each subdomain) that can be solved simultaneously.

The method is set for a general nonlinear problem in several space variables. The convergence analysis, however, is carried out only for a linear one-dimensional system with continuous solutions. A precise form of the error-reduction factor at each iteration is derived.

Although the method is applied here to the case of spectral collocation approximation only, the idea is fairly general and can be used in a different context as well. For instance, its application to space discretization by finite differences is straightforward.

Key words. hyperbolic systems, domain decomposition, spectral collocation, convergence analysis

AMS(MOS) subject classifications. 65N35, 65N15, 35L50

Introduction. In this paper we propose an effective domain decomposition method for the numerical solution of quasi-linear hyperbolic systems of conservation laws.

Although our emphasis is on spatial discretization by spectral collocation schemes, the domain decomposition approach we advocate here can be adopted for finite difference discretizations as well. At every time-level the method applies after the initial value problem has been advanced by either an implicit or an explicit timestepping.

The spatial computational domain is subdivided into several adjoining, nonintersecting subdomains; within each of them we look for a polynomial solution of degree N with respect to each component. If Γ_i denotes the interface between two of such subdomains, say Ω_i and Ω_{i+1}, at each gridpoint on Γ_i we enforce the conditions of continuity of the physical variables. Furthermore, we require the fulfillment of the so-called compatibility equations, i.e., of those characteristic combinations of the original equations that express wave propagation across Γ_i from Ω_i toward Ω_{i+1} and vice versa.

The main reason for using the multidomain spectral method rather than the standard single-domain spectral approach stems from its capability of covering problems in complex geometry. Moreover, the spectral multidomain approach allows local refinement to resolve internal layers (or even discontinuities) maintaining however the spectral accuracy enjoyed by the classical spectral collocation method (e.g., [4, Chap. 12]).

We also propose here a new iteration-by-subdomain algorithm that allows the decoupling of the subproblems arising from the multidomain approach making it possible to solve at each iteration as many independent subproblems as the number of subdomains. This algorithm requires that on Γ_i the values of the characteristic variables impinging Ω_i equate those outgoing from Ω_{i+1} at the previous iteration, and vice versa.

For the case of linear hyperbolic systems we prove that the above iteration-by-subdomain method is convergent. Furthermore, we find a close expression of the

* Received by the editors January 11, 1989; accepted for publication (in revised form) August 20, 1989.

† Dipartimento di Matematica, Università Cattolica, Via Trieste 17, 25121 Brescia, Italy; and Istituto di Analisi Numerica del Consiglio Nazionale delle Ricerche, C.so Carlo Alberto 5, 27100 Pavia, Italy. Present address, Dipartimento di Matematica, Politecnico di Milano, Via Bonardi 9, 20133 Milano, Italy.

reduction faction per iteration and show that it is independent of the number of gridpoints inside each subdomain. Furthermore, an algebraic interpretation of our iteration-by-subdomain algorithm is derived in terms of the Schur complement (or capacitance) matrix.

Finally, we show how the method proposed here can be adapted to the case in which an internal interface is a discontinuity surface (actually, a "k-shock" in the sense of Lax [10]). This typically occurs if a shock fitting approach is adopted [4, § 8.6]. In such a case, the compatibility equations together with the Rankine–Hugoniot conditions allow us to compute first the flow field within the upstream subdomain then the one in the downstream subdomain.

When a shock capturing strategy is pursued, domain decomposition makes it possible to use numerical viscosity only within the (thin) subdomain embodying the shock front. In this case, the subdomain interfaces are dealt with as continuity lines (or surfaces).

Examples of domain decomposition methods of similar type have been proposed in the latest years. We recall for instance [9] and the references given in Chapter 13 of [4]. We also refer to [11] and [12] for applications of spectral multidomain techniques to reacting flow whose shape and motion are generated at each time-level. In the frame of finite differences we refer to the earlier paper [3] where the issue of the compatibility equations at subdomain interfaces was addressed. More recent applications of finite difference multidomain methods for compressible flow simulations can be found in [2].

The present one is, to the author's knowledge, the first paper in which the matching and compatibility conditions at subdomain interfaces are properly used and fully justified on a theoretical ground. An outline of this paper is reported below. In § 1 we introduce the initial boundary value problem and the compatibility equations, and in § 2 we state the domain decomposition formulation of the Chebyshev collocation approximation to the problem. In § 3 we present an iteration-by-subdomain algorithm for an effective solution of the domain decomposition problem. Thereafter we confine ourselves to the case of one-dimensional hyperbolic systems.

The interest of illustrating our method on one-dimensional problems is twofold. First, the simple structure of the problem allows a better understanding of our iteration-by-subdomain method, whose presentation within the general framework of previous sections might appear fairly involved. Second, the one-dimensional case is the only one in which we can carry out a sound convergence analysis. Indeed, for one-dimensional wave equations we are able to find the close form of the spectral collocation solution.

In § 4 we write the domain decomposition problem as well as the iteration-by-subdomain algorithms. In § 5 we carry out the convergence analysis for the iterative algorithm, and in § 6 we derive the relationship with the Schur complement matrix associated to the interface unknowns. Finally, in § 7, we report some numerical results for a one-dimensional model problem.

1. The hyperbolic system and the compatibility equations.

We consider the following differential system:

$$\frac{\partial u}{\partial t}+\sum_{j=1}^{m} A_j(u)\frac{\partial u}{\partial x_j}=f \quad \text{in } \Omega\times(0, T) \tag{1.1}$$

where $m=1$, 2, or 3, Ω is an open bounded domain of $\mathbb{R}^m$; u, and f are two vector functions $u, f:\Omega\times(0, T)\rightarrow\mathbb{R}^p$, with $p\geqq 1$; and $A_j(u)$ are $p\times p$ matrices possibly depending on u.

The system (1.1) is supplemented by initial conditions of the form

$$u(x,0)=\varphi(x) \quad \forall x\in\Omega \tag{1.2}$$

and by suitable boundary conditions at $(0,T)\times\partial\Omega$ that will be specified later.

For any $\xi\in\mathbb{R}^m$ such that $|\xi|=(\sum_{j=1}^m \xi_j^2)^{1/2}=1$, let us define the characteristic matrix for the ξ direction as follows:

$$A(\xi):=\sum_{j=1}^m A_j(u)\xi_j. \tag{1.3}$$

The system (1.1) is assumed to be hyperbolic in time: this means that for any such ξ, $A(\xi)$ has p real eigenvalues and moreover it is diagonalizable.

Let us denote by $\{\lambda^k, k=1,\cdots,p\}$ the eigenvalues of $A(\xi)$, and by $\{v^k, k=1,\cdots,p\}$ the set of the corresponding left eigenvectors, so that

$$v^kA(\xi)=\lambda^kv^k, \qquad k=1,\cdots,p. \tag{1.4}$$

We assume that $\lambda^k>0$ for $k=1,\cdots,q$ and $\lambda^k<0$ for $k=q+1,\cdots,p$ for some $0\leqq q\leqq p$ (obviously, q depends on the particular direction ξ that we are considering). Let us take the inner product of v^k with (1.1); for each k we obtain the scalar equation

$$v^k\cdot\left(\frac{\partial u}{\partial t}+\sum_{j=1}^m A_j(u)\frac{\partial u}{\partial x_j}\right)=v^k\cdot f, \qquad k=1,\cdots,p. \tag{1.5}$$

Now denote by $(\tau_1,\cdots,\tau_{m-1})$ the system of Cartesian coordinates of the hyperplane orthogonal to the direction ξ. Then for each $h=1,\cdots,m-1$, $\tau_h=(\tau_{h_1},\cdots,\tau_{h_m})^T\in\mathbb{R}^m$, $\tau_h\cdot\xi=0$ and $|\tau_h|=1$. Owing to the identity

$$\frac{\partial u}{\partial x_j}=\frac{\partial u}{\partial \xi}\xi_j+\sum_{h=1}^{m-1}\frac{\partial u}{\partial \tau_h}\tau_{h_j}, \qquad j=1,\cdots,m \tag{1.6}$$

from (1.5) and (1.4) it follows that

$$v^k\cdot\left(\frac{\partial u}{\partial t}+\lambda^k\frac{\partial u}{\partial \xi}\right)=v^k\cdot\left(f-\sum_{j=1}^m A_j(u)\sum_{h=1}^{m-1}\frac{\partial u}{\partial \tau_h}\tau_{h_j}\right), \qquad k=1,\cdots,p. \tag{1.7}$$

Following a terminology proposed in [3], we will refer to (1.7) as the *compatibility equations* for the problem (1.1).

Remark 1.1. Let Γ denote a $(m-1)$-dimensional manifold of $\mathbb{R}^m$, and denote by ν the unit normal direction to Γ. Taking $\xi=\nu$, equations (1.7) restricted to Γ become

$$v^k\cdot\left(\frac{\partial u}{\partial t}+\lambda^k\frac{\partial u}{\partial \nu}\right)=v^k\cdot\left(f-\sum_{j=1}^m A_j(u)\sum_{h=1}^{m-1}\frac{\partial u}{\partial \tau_h}\tau_{h_j}\right), \qquad k=1,\cdots,p. \tag{1.8}$$

The right-hand side of (1.8) depends on the tangential derivatives of u on Γ, whereas the left-hand side yields a combination (through the components of the eigenvector v^k) of transport equations along a direction that is normal to Γ.

Assume Γ is the boundary of Ω and ν is oriented outward Ω (see Fig. 1.1(a)); then for $k=1,\cdots,q$, (1.8) yield q transport equations according to which information are propagated from the inside to the outside of Ω. For such a reason, equations (1.8) for $k=1,\cdots,q$ will be called the compatibility equations *for the domain* Ω.

If we now assume that Ω_1 and Ω_2 are two adjoining subdomains of Ω, and Γ is their common boundary, taking as ν the normal direction to Γ, oriented from Ω_1 to Ω_2 (see Fig. 1.1(b)), then the first q equations of (1.8) are the compatibility equations *for* Ω_1 (they entail propagation of information from Ω_1 to Ω_2). Obviously, for $k=q+1,\cdots,p$, (1.8) provides the compatibility equations *for* Ω_2. Such a distinction will

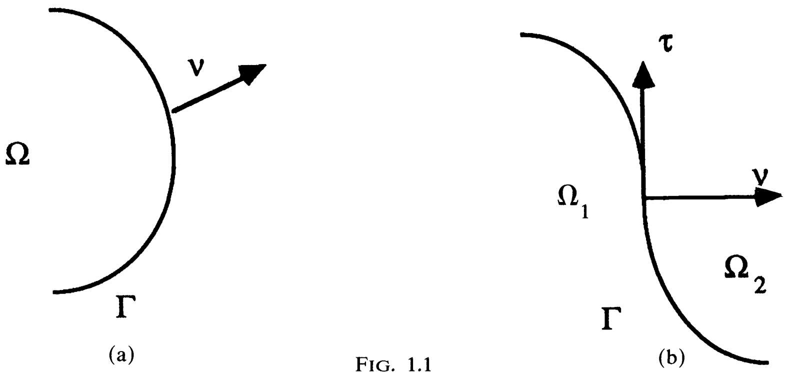


FIG. 1.1

be crucial in the definition of the multidomain method we will propose in the forthcoming sections. □

2. Spectral collocation approximation to the system (1.1): A domain decomposition approach. In this section we assume that Ω is an m-dimensional hypercube. Let Ω_1 and Ω_2 be two open subregions so that $\bar{\Omega}=\bar{\Omega}_1\cup\bar{\Omega}_2$, $\bar{\Omega}_1\cap\bar{\Omega}_2=\Gamma$, $\Omega_1\cap\Omega_2=\varnothing$ where Γ is orthogonal to one Cartesian direction. We assume here that the solution of (1.1) is continuous across Γ. The case of solutions that are discontinuous across Γ is faced in § 2.1.

Let N be a given positive integer; we denote by

$$\Sigma := \left\{ x_{\mathbf{k}} = \cos\frac{\pi \mathbf{k}}{N}, \mathbf{k}=(k_1,\cdots,k_m), 0\leqq k_i \leqq N, i=1,\cdots,m \right\} \tag{2.1}$$

the set of Chebyshev-Lobatto points of the reference hypercube $[-1,1]^m$. Furthermore, we denote by Σ^1 and Σ^2 the corresponding set of points in Ω_1 and Ω_2, respectively. Note that a point of Σ^1 (respectively, Σ^2) lies on the boundary of Ω_1 (respectively, Ω_2) if at least one of its subindices k_j is either zero or N. Since we are using the same number of points in Ω_1 and Ω_2, the points of $\Sigma^1\cap\Gamma$ and those of $\Sigma^2\cap\Gamma$ are exactly the same. We will denote by Σ_Γ this common set of interface points. Furthermore, for each $k=1, 2$, we denote by Σ_0^k the points of Σ^k that are internal to Ω_k, i.e., which do not lie on $\partial\Omega_k$.

A (continuous in time) spectral domain decomposition method for problem (1.1) is defined as follows. At each time $t>0$ we look for $u_N^i(t)\in(\mathbb{P}_N(\Omega_i))^p$ that satisfies the following set of equations.

(a) At each point of Σ_0^1

$$\frac{\partial u_N^1}{\partial t}+\sum_{j=1}^m A_j(u_N^1)\frac{\partial u_N^1}{\partial x_j}=f; \tag{2.2}$$

(b) At each point of Σ_0^2

$$\frac{\partial u_N^2}{\partial t}+\sum_{j=1}^m A_j(u_N^2)\frac{\partial u_N^2}{\partial x_j}=f; \tag{2.3}$$

(c) At each point of Σ_Γ

$$u_N^1 = u_N^2. \tag{2.4}$$

Moreover, we require that

$$v^k \cdot \left(\frac{\partial u_N^1}{\partial t} + \lambda^k \frac{\partial u_N^1}{\partial \nu} \right) = v^k \cdot \left(f - \sum_{j=1}^{m} A_j(u_N^1) \sum_{h=1}^{m-1} \frac{\partial u_N^1}{\partial \tau_h} \tau_{h_j} \right), \qquad k = 1, \cdots, q. \tag{2.5}$$

ν is the outward unit normal direction to Ω_1 on Γ (see Fig. 1.1(b)), λ^k and v^k are the eigenvalues and the (left) eigenvectors of the matrix $A(\nu)$ (with $\lambda^k > 0$ for $k = 1, \cdots, q$), and τ is the tangential direction on Γ.

Furthermore, we satisfy

$$v^k \cdot \left(\frac{\partial u_N^2}{\partial t} + \lambda^k \frac{\partial u_N^2}{\partial \nu} \right) = v^k \cdot \left(f - \sum_{j=1}^{m} A_j(u_N^2) \sum_{h=1}^{m-1} \frac{\partial u_N^2}{\partial \tau_h} \tau_{h_j} \right), \qquad k = q+1, \cdots, p. \tag{2.6}$$

Following the definitions given in Remark 1.1, (2.5) are the q compatibility equations for Ω_1 on Γ, while (by virtue of (2.4)), (2.6) are the $p-q$ ones for Ω_2.

(d) At each point of Σ^1 belonging to a "face" Φ of $\partial\Omega_1$ (excluding the interface Γ) we set

$$v^k \cdot \left(\frac{\partial u_N^1}{\partial t} + \lambda^k \frac{\partial u_N^1}{\partial \nu} \right) = v^k \cdot \left(f - \sum_{j=1}^{m} A_j(u_N^1) \sum_{h=1}^{m-1} \frac{\partial u_N^1}{\partial \tau_h} \tau_{h_j} \right), \qquad k = 1, \cdots, q \tag{2.7}$$

where ν is now the outward normal direction to Ω_1 on Φ, τ is the tangential direction on Φ, while λ^k and v^k are the eigenvalues and the (left) eigenvectors of the matrix $A(\nu)$ (note that here q is not necessarily the same of (2.5)).

The remaining $p-q$ equations that we need at each collocation point of Φ must be provided by the physical boundary conditions that supplement (1.1) and (1.2).

(e) At each point of Σ^2 belonging to a "face" Φ of $\partial\Omega_2$ (excluding the interface Γ) we enforce

$$v^k \cdot \left(\frac{\partial u_N^2}{\partial t} + \lambda^k \frac{\partial u_N^2}{\partial \nu} \right) = v^k \cdot \left(f - \sum_{j=1}^{m} A_j(u_N^2) \sum_{h=1}^{m-1} \frac{\partial u_N^2}{\partial \tau_h} \tau_{h_j} \right), \qquad k = 1, \cdots, q \tag{2.8}$$

where ν is the outward normal direction to Ω_2 on Φ, τ the tangential direction on Γ, λ^k, and v^k are the eigenvalues and the (left) eigenvectors of the matrix $A(\nu)$. (Again, $\lambda^k > 0$ for $k = 1, \cdots, q$, where q here is not necessarily the same as before.)

The remaining $p-q$ equations are prescribed as boundary conditions.

Remark 2.1. The use of the compatibility equations (2.5)-(2.8) requires the knowledge of eigenvalues and left eigenvectors of $A(\nu)$. If the matrices A_j are not constant, by elementary symbolic calculation on the $p \times p$ matrix $A(\nu)$, eigenvalues as well as eigenvectors can be generally expressed in close form in terms of the components of the physical variable and of ν. Their values at each iteration and at any point of Γ can therefore be obtained by saturating the dependency on specific available values of the solution. □

Remark 2.2. In the frame of classical *single-domain* spectral collocation methods for hyperbolic systems, the use of compatibility equations for collocation *boundary* points was first advocated in [8] and [5]. For the same method, a stability and convergence analysis were developed in [6] and [7] for the case of dissipative boundary conditions. No convergence analysis (with respect to N) is available so far for spectral multidomain approximations of hyperbolic systems. □

Remark 2.3 (fully discrete approximation). A fully discrete approximation to the initial value problem (1.1), (1.2) can be obtained using a time marching scheme in (2.2)-(2.8). Whatever scheme (either implicit or explicit) we adopt to advance from a known time level t^k to a new one t^{k+1}, the matching interface condition (2.4), as well as the prescribed boundary conditions should be enforced at the new time level.

If an *explicit* timestepping is used, at the time level t^{k+1} the unknown vectors $\{\mathbf{u}_N^1(x_j^1), x_j^1 \in \Sigma_0^1\}$ and $\{\mathbf{u}_N^2(x_j^2), x_j^2 \in \Sigma_0^2\}$, can be computed using solely the internal equations (2.2) and (2.3), respectively. Once these internal values are available, the interface equations (2.4)-(2.6), together with the boundary equations (2.7), (2.8), and the prescribed boundary conditions can be solved to provide the remaining values of the unknowns. Actually, we note that the presence of derivatives in space among boundary and interface equations relates boundary and interface values to each other. We also emphasize that the differential equations (2.5)-(2.8) should be advanced by the same explicit timestepping that was used for the equations at the internal points.

When an *implicit* timestepping is used, the internal unknowns are no longer decoupled from the remaining ones. We will see an example in the next section. □

2.1. The case of discontinuous solutions across the subdomain interface. We now consider the case where Γ is no longer an arbitrary surface, but rather the front of a shock propagating throughout the computational domain Ω. This case typically occurs when a shock fitting technique is adopted with the purpose of determining the shape and the motion law of the shock front at each time level (see, e.g., [4, § 8.6]). We still denote by Ω_1 and Ω_2 the adjoining subdomains separated by Γ, by ν the outward normal vector to Ω_1 on Γ and with w the speed with which Γ propagates in the direction ν.

We will assume that there exists a vector function $F:\Omega\times(0,T)\to\mathbb{R}^p$ such that $A_j(u)=\partial F_j(u)/\partial u$ so that (1.1) can be written in conservation form as follows:

$$\frac{\partial u}{\partial t}+\sum_{j=1}^{m}\frac{\partial F_j(u)}{\partial x_j}=f \quad \text{in } \Omega\times(0,T). \tag{2.9}$$

The weak solutions to (2.9) (e.g., [10]) are allowed to be discontinuous across the interface Γ, but should satisfy the following jump conditions (*Rankine-Hugoniot conditions*):

$$w[u]-\sum_{j=1}^{m}\nu_j[F_j(u)]=0. \tag{2.10}$$

Here $[\cdot]$ denotes the difference between values in the brackets on the two sides of Γ. In the context of the above spectral Chebyshev approximation, the above equations read as follows:

$$w(u_N^2-u_N^1)-\sum_{j=1}^{m}\nu_j\{F_j(u_N^2)-F_j(u_N^1)\}=0. \tag{2.11}$$

At each collocation point Σ_Γ on Γ, (2.11) yields p equations for the $2p+1$ unknowns: u_N^1, u_N^2 and w. We need therefore $p+1$ further independent conditions that will be provided by the compatibility equations, which, in the current situations, can be determined as follows. Let us define the characteristic matrix for Ω_1 at the point $P\in\Gamma$:

$$A(\nu^1)=\sum_{j=1}^{m}\nu_j A_j(u_N^1) \tag{2.12}$$

and denote by $\lambda_{(1)}^k$ and $v_{(1)}^k$ the eigenvalues and left eigenvectors, respectively, of $A(\nu^1)$. Similarly, let us denote by $\lambda_{(2)}^k$ and $v_{(2)}^k$, respectively, the eigenvalues and left eigenvectors of the matrix

$$A(\nu^2)=\sum_{j=1}^{m}(-\nu_j)A_j(u_N^2) \tag{2.13}$$

(note that this time $\lambda_{(1)}^k \neq -\lambda_{(2)}^k$ and $v_{(1)}^k \neq v_{(2)}^k$ in general, since u_N^1 and u_N^2 are no longer coincident on Γ). Assume that the (real) eigenvalues are ordered as follows:

$$\lambda_{(1)}^j<\lambda_{(1)}^{j+1}, \quad \lambda_{(2)}^i<\lambda_{(2)}^{i+1}, \quad i,j=1,\cdots,p-1.$$

We assume that Γ is the surface of propagation of a *k-shock*, so that the following entropy conditions are verified (e.g., [13, p. 261]). There exists an integer $k\in\{1,\cdots,p\}$ such that

$$\lambda_{(1)}^{k-1}<w<\lambda_{(1)}^{k}, \qquad \lambda_{(2)}^{k}<w<\lambda_{(2)}^{k+1}. \tag{2.14}$$

Figure 2.1 illustrates an example of a k-shock for a one-dimensional problem. For a fixed time t we have drawn in bold the straight line with slope $=w$ (the speed of the shock front). On its left (respectively, right) we have labeled with j the straight line whose slope is equal to the characteristic speed $\lambda_1^{(j)}$ (respectively, $\lambda_2^{(j)}$), for $j=1,\cdots,p$.

The compatibility equations for Ω_1 will therefore be given by the $p-k+1$ equations

$$v_{(1)}^r\cdot\left(\frac{\partial u_N^1}{\partial t}+\lambda_{(1)}^r\frac{\partial u_N^1}{\partial \nu^1}\right)=v_{(1)}^r\cdot\left(f-\sum_{j=1}^{m}A_j(u_N^1)\sum_{h=1}^{m-1}\frac{\partial u_N^1}{\partial \tau_h}\tau_{h_j}\right), \qquad r=k,\cdots,p, \tag{2.15}$$

whereas those for Ω_2 are given by the k equations

$$v_{(2)}^s\cdot\left(\frac{\partial u_N^2}{\partial t}+\lambda_{(2)}^s\frac{\partial u_N^2}{\partial \nu^2}\right)=v_{(2)}^s\cdot\left(f-\sum_{j=1}^{m}A_j(u_N^2)\sum_{h=1}^{m-1}\frac{\partial u_N^2}{\partial \tau_h}\tau_{h_j}\right), \tag{2.16}$$

$$s=p-k+1,\cdots,p.$$

As usual, $\{\tau_h\}$ is the set of vectors tangential to Γ at the point P under consideration.

Note that (2.14) can be rewritten in the form (see [10, p. 25])

$$\lambda_{(1)}^{k-1}<w<\lambda_{(2)}^{k+1}, \qquad \lambda_{(2)}^{k}<w<\lambda_{(1)}^{k}, \tag{2.17}$$

which shows that there exists only one index k such that the shock speed w is intermediate to the characteristic speeds λ^k on both sides of the shock.

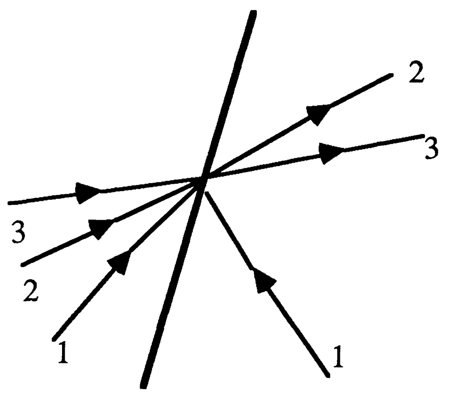


FIG. 2.1. *A k-shock with $p=3$ and $k=1$.*

In the case of a flow regime supersonic in Ω_1 (upstream subdomain) and subsonic in Ω_2 (downstream subdomain) (see, e.g., Fig. 2.1) the flow field within Ω_1 is uninfluenced by that within Ω_2, and therefore should be computed first.

Remark 2.4 (discontinuity due to unsmooth initial data). For linear hyperbolic systems (A_j independent of u in (1.1)) with discontinuous initial data, there is propagation of discontinuity across fronts whose position and motion can be a priori determined in terms of the data. In these cases, w is given and $\lambda^k_{(1)} = -\lambda^{(k)}_{(2)}$, $v^{(k)}_{(1)} = v^{(k)}_{(2)}$ for all $k = 1, \cdots, p$, so that the p resulting compatibility equations together with the p Rankine–Hugoniot conditions (2.11) allow the calculation of the $2p$ unknowns u^1_N, u^2_N at each collocation point on the discontinuity fronts. □

3. An iteration-by-subdomain algorithm for the solution of the domain decomposition problem. Let us return to the case of solutions that are continuous across the subdomain interface Γ. The description of the domain decomposition method given in the previous section shows that the spectral collocation problems in Ω_1 and Ω_2 are coupled throughout the continuity conditions (2.4) at the interface Γ. To remove this coupling we propose the following iterative method. Assume the solution is available at the nth step. Let ν, τ, λ^k, and v^k be defined as at the point (c) of § 2 for each collocation point of Σ_Γ (clearly, λ^k and v^k depend on the value of the solution at the points of Γ).

Let $\wedge(\nu)$ and $T(\nu)$ denote, respectively, the matrix of the eigenvalues of $A(\nu)$ and that of the corresponding (left) eigenvectors, so that

$$T(\nu)A(\nu) = \wedge(\nu)T(\nu). \tag{3.1}$$

Then define at the point under consideration

$$\chi^1 = (u^1_N)^n_{|\Gamma}, \qquad \chi^2 = (u^2_N)^n_{|\Gamma}. \tag{3.2}$$

Finally, we denote by $T_q(\nu)$ the $q \times p$ matrix given by the first q rows of $T(\nu)$, whereas $T_{p-q}(\nu)$ will denote the $(p-q) \times q$ matrix given by the remaining $p-q$ rows of $T(\nu)$. The solutions at the new step, say $(u^1_N)^{n+1}$ in Ω_1 and $(u^2_N)^{n+1}$ in Ω_2, can be obtained by solving two independent problems in Ω_1 and in Ω_2, respectively.

Precisely, $(u^1_N)^{n+1}$ satisfies the interior equations (2.2), the interface equations (2.5) together with

$$T_{p-q}(\nu)(u^1_N)^{n+1} = T_{p-q}(\nu)\chi^2 \quad \text{on } \Gamma \tag{3.3}$$

and, finally, the boundary equations given at the point (d) of § 2.

Similarly, $(u^2_N)^{n+1}$ satisfies the interior equations (2.3), the interface equations (2.6) together with

$$T_q(\nu)(u^2_N)^{n+1} = T_q(\nu)\chi^1 \quad \text{on } \Gamma \tag{3.4}$$

and, finally, the boundary equations prescribed at the point (e) of § 2. Note that the limit solutions $u^1_N = \lim_{n\to\infty} (u^1_N)^n$, $u^2_N = \lim_{n\to\infty} (u^2_N)$ satisfy the conditions

$$T_q(\nu)u^2_N = T_q(\nu)u^1_N \quad \text{on } \Gamma,$$

$$T_{p-q}(\nu)u^1_N = T_{p-q}(\nu)u^2_N \quad \text{on } \Gamma,$$

which in turn ensure the fulfillment of the continuity requirement (2.4).

Remark 3.1. The extension of the above method to decompositions with several subdomains is straightforward. At each step, we are left to solve as many independent subproblems as the number of subdomains. All these subproblems can be solved simultaneously within a parallel computer environment. □

4. A particular case: One-dimensional problems. We consider the special case in which Ω is the one-dimensional interval $(-1, 1)$, and $\Omega_1 = (-1, \alpha)$, $\Omega_2 = (\alpha, 1)$ for some $-1 < \alpha < 1$. In this case $\Gamma = \{\alpha\}$.

The differential system reads as follows:

$$\frac{\partial u}{\partial t} + A(u)\frac{\partial u}{\partial x} = f \quad \text{in } \Omega \times (0, T). \tag{4.1}$$

Denoting with $\{\lambda^k, k = 1, \cdots, p\}$ and $\{v^k, k = 1, \cdots, p\}$ the eigenvalues and the left eigenvectors of $A(u)$, respectively, the compatibility equations (1.7) in the current case become

$$v^k \cdot \left(\frac{\partial u}{\partial t} + \lambda^k \frac{\partial u}{\partial x}\right) = v^k \cdot f, \qquad k = 1, \cdots, p. \tag{4.2}$$

Setting $\wedge = \operatorname{diag}\{\lambda^1, \cdots, \lambda^p\}$ and denoting by T the matrix whose kth row contains v^k, we can write (4.2) as follows:

$$T\frac{\partial u}{\partial t} + \wedge T\frac{\partial u}{\partial x} = Tf. \tag{4.3}$$

Obviously, T and $\wedge$ depend on u if so does A.

Assume that at the interface point $x = \alpha$ the first q eigenvalues of A are positive. (Of course, A, $\wedge$, T and q depend on the time level t.) Let us denote by $\wedge_q$ and T_q the $q \times p$ matrices obtained suppressing the last $p - q$ rows of $\wedge$ and T, respectively. Then the following q equations

$$T_q\frac{\partial u}{\partial t} + \wedge_q T\frac{\partial u}{\partial x} = T_q f \quad \text{at } x = \alpha \tag{4.4}$$

provide the q compatibility equations for Ω_1. Similarly, denoting with $\wedge_{p-q}$ and T_{p-q} the lower part of the matrix $\wedge$ and T, respectively, the equations

$$T_{p-q}\frac{\partial u}{\partial t} + \wedge_{p-q} T\frac{\partial u}{\partial x} = T_{p-q} f \quad \text{at } x = \alpha \tag{4.5}$$

provide the $p - q$ compatibility equations for Ω_2. The compatibility equations for Ω_2 at the right-hand boundary point $x = 1$ are obtained in a similar way, and take the form (4.4), whereas at the left-hand boundary $x = -1$ the compatibility equations for Ω_1 take the form (4.5).

The hyperbolic system (4.1) needs to be completed by the initial condition

$$u(x, 0) = \varphi(x), \qquad x \in \Omega \tag{4.6}$$

and by a set of boundary conditions (q equations at the point $x = -1$, and $p - q$ at the point $x = 1$) that we assume have the form

$$\begin{aligned} B_1 u &= g^1 \quad \text{at } x = -1, \\ B_2 u &= g^2 \quad \text{at } x = 1 \end{aligned} \tag{4.7}$$

where B_1 is a $q \times p$ matrix, B_2 is a $(p - q) \times p$ matrix, and g^1 and g^2 are two given vector functions.

Remark 4.1. Since $\wedge T = TA$ from (3.1), instead of (4.3), we can write

$$T\left[\frac{\partial u}{\partial t} + A\frac{\partial u}{\partial x}\right] = Tf. \tag{4.3$'$}$$

This is exactly the form that was used in [5] to derive the numerical boundary conditions at the physical boundary of the domain.

We also note that if A is a constant matrix, then setting $z = Tu$ (characteristic variables), from (4.3) we obtain the following characteristic form of the compatibility equations:

$$\frac{\partial z}{\partial t} + \Lambda \frac{\partial z}{\partial x} = Tf. \tag{4.3$''$}$$ □

We will now apply the multidomain spectral collocation method described in § 2 to the one-dimensional problem (4.1). Let us define

$$x_j^1 = -1 + \frac{\alpha+1}{2}(t_j+1), \quad x_j^2 = \alpha + \frac{1-\alpha}{2}(t_j+1), \quad j = 0, \cdots, N \quad \text{where } t_j = \cos\frac{\pi j}{N}.$$

The points t_j are the Chebyshev-Lobatto nodes in the reference interval $[-1, 1]$, whereas x_j^1 and x_j^2 are their images within the subdomains Ω_1 and Ω_2, respectively (note that $-1 < x_j^1 < \alpha < x_j^2 < 1$ for $1 \leqq j \leqq N-1$).

At each time $t > 0$ we look for $u_N^1(t) \in (\mathbb{P}_N(\Omega_1))^p$, $u_N^2(t) \in (\mathbb{P}_N(\Omega_2))^p$ satisfying the following:

(a) Internal equations

$$\frac{\partial u_N^1}{\partial t} + A(u_N^1)\frac{\partial u_N^1}{\partial x} = f \quad \text{at } x_j^1, \quad 1 \leqq j \leqq N-1, \tag{4.8}$$

$$\frac{\partial u_N^2}{\partial t} + A(u_N^2)\frac{\partial u_N^2}{\partial x} = f \quad \text{at } x_j^2, \quad 1 \leqq j \leqq N-1; \tag{4.9}$$

(b) Interface equations

$$T_q\frac{\partial u_N^1}{\partial t} + \Lambda_q T\frac{\partial u_N^1}{\partial x} = T_q f \quad \text{at } x = \alpha, \tag{4.10}$$

$$T_{p-q}\frac{\partial u_N^2}{\partial t} + \Lambda_{p-q} T\frac{\partial u_N^2}{\partial x} = T_{p-q} f \quad \text{at } x = \alpha, \tag{4.11}$$

$$u_N^1 = u_N^2 \quad \text{at } x = \alpha; \tag{4.12}$$

(c) Boundary equations

$$T_q\frac{\partial u_N^2}{\partial t} + \Lambda_q T\frac{\partial u_N^2}{\partial x} = T_q f \quad \text{at } x = 1, \tag{4.13}$$

$$T_{p-q}\frac{\partial u_N^1}{\partial t} + \Lambda_{p-q} T\frac{\partial u_N^1}{\partial x} = T_{p-q} f \quad \text{at } x = -1, \tag{4.14}$$

$$B_2 u_N^2 = g^2 \quad \text{at } x = 1, \tag{4.15}$$

$$B_1 u_N^1 = g^1 \quad \text{at } x = -1. \tag{4.16}$$

Note that (4.10) and (4.14) are the compatibility equations for Ω_1 at the points $x = \alpha$ and $x = -1$, respectively, whereas (4.11) and (4.13) are those of Ω_2 at the points $x = \alpha$ and $x = 1$, respectively. Finally, (4.15) and (4.16) are the boundary conditions (see (4.7)).

We now write the *iteration-by-subdomain method* described in § 3 for the solution to the problem (4.8)-(4.16). Assume $(u_N^1)^n$, $(u_N^2)^n$ are available at the nth step and define

$$\chi^1 = (u_N^1)^n, \quad \chi^2 = (u_N^2)^n \quad \text{at } x = \alpha. \tag{4.17}$$

Then in Ω_1 we look for $(u_N^1)^{n+1}(t) \in (\mathbb{P}_N(\Omega_1))^p$ such that

$$\frac{\partial}{\partial t}(u_N^1)^{n+1} + A\frac{\partial}{\partial x}(u_N^1)^{n+1} = f \quad \text{at } x_j^1, \quad 1 \leqq j \leqq N-1, \tag{4.18}$$

$$T_q\frac{\partial}{\partial t}(u_N^1)^{n+1} + \wedge_q T\frac{\partial}{\partial x}(u_N^1)^{n+1} = T_q f \quad \text{at } x = \alpha, \tag{4.19}$$

$$T_{p-q}(u_N^1)^{n+1} = T_{p-q}\chi^2 \quad \text{at } x = \alpha, \tag{4.20}$$

$$T_{p-q}\frac{\partial}{\partial t}(u_N^1)^{n+1} + \wedge_{p-q} T\frac{\partial}{\partial x}(u_N^1)^{n+1} = T_{p-q} f \quad \text{at } x = -1, \tag{4.21}$$

$$B_1(u_N^1)^{n+1} = g^1 \quad \text{at } x = -1 \tag{4.22}$$

where the matrices A, $\wedge$, and T depend on $(u_N^1)^{n+1}$.

In Ω_2 we solve for $(u_N^2)^{n+1}(t) \in (\mathbb{P}_N(\Omega_2))^p$ satisfying

$$\frac{\partial}{\partial t}(u_N^2)^{n+1} + A\frac{\partial}{\partial x}(u_N^2)^{n+1} = f \quad \text{at } x_j^2, \quad 1 \leqq j \leqq N-1, \tag{4.23}$$

$$T_{p-q}\frac{\partial}{\partial t}(u_N^2)^{n+1} + \wedge_{p-q} T\frac{\partial}{\partial x}(u_N^2)^{n+1} = T_{p-q} f \quad \text{at } x = \alpha, \tag{4.24}$$

$$T_q(u_N^2)^{n+1} = T_q\chi^1 \quad \text{at } x = \alpha, \tag{4.25}$$

$$T_q\frac{\partial}{\partial t}(u_N^2)^{n+1} + \wedge_q T\frac{\partial}{\partial x}(u_N^2)^{n+1} = T_q f \quad \text{at } x = 1, \tag{4.26}$$

$$B_2(u_N^2)^{n+1} = g^2 \quad \text{at } x = 1. \tag{4.27}$$

In (4.23)-(4.26) the matrices A, $\wedge$, and T now depend on $(u_N^2)^{n+1}$.

Note that the problem in Ω_1 is independent of that in Ω_2.

The extension of the above iteration-by-subdomain method to the case of a subdivision by M subdomains ($M > 2$) is straightforward. At each iteration we obtain M independent subproblems that can be solved simultaneously.

Here above, the iteration-by-subdomain method has been applied to the semidiscrete (continuous in time) problem (4.8)-(4.16). To be effective, the iteration method should be applied at each timestep after which a time marching scheme has been used to get a full space-time discretization of the problem (see the previous remark). In this way, at the new time level t^{k+1}, we can apply the iterative method until we achieve convergence to the spectral multidomain solutions $u_N^1(t^{k+1})$, $u_N^2(t^{k+1})$.

5. Convergence analysis for the iteration-by-subdomain method. In this section we will prove that the iteration-by-subdomain method introduced so far is convergent. Our analysis is confined to one-dimensional hyperbolic systems with constant matrix. The initial boundary value problem under investigation takes the form (4.1), (4.6), (4.7) where A is a 2×2 matrix that is not restrictive to assume the following form:

$$A = \begin{pmatrix} a & 1 \\ 1 & a \end{pmatrix} \quad \text{with } |a| < 1. \tag{5.1}$$

The eigenvalues of A, say λ and $-\mu$, have opposite sign, as $\lambda = a+1 > 0$ and $-\mu = a-1 < 0$. We note that in this case

$$A = T \wedge T, \text{ where } \wedge = \text{diag}\{\lambda, -\mu\} \text{ and } T = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{5.2}$$

Among the various sets of boundary conditions that render this problem well posed, we consider the following:

(5.3) $$u_1(-1,t)=0, \quad u_1(1,t)=0 \quad \forall t\in(0,T).$$

We will require the initial value $\varphi(x)$ (see (1.2)) to be a continuous vector function, with a first component vanishing at $x=-1$ and $x=1$. Under this assumption, initial and boundary conditions are compatible, hence the solution to our problem is continuous for all time. We assume that the spectral multidomain problem (4.8)-(4.16) is advanced in time by an implicit method. For the sake of simplicity, let us consider the backward Euler method (no essential modification occurs if considering a different implicit method). When we advance from t^k to t^{k+1}, at the new level t^{k+1} the new functions $u^1:=(u_N^1)(t^{k+1})$ and $u^2:=(u_N^2)(t^{k+1})$ satisfy the following multidomain problem:

(5.4) $$\beta u^1+Au_x^1=f_{\text{prev}}^1 \quad \text{at } x_j^1, \quad 1\leqq j\leqq N-1,$$

(5.5) $$\beta u^2+Au_x^2=f_{\text{prev}}^2 \quad \text{at } x_j^2, \quad 1\leqq j\leqq N-1,$$

(5.6) $$T_q\beta u^1+\lambda T_q u_x^1=T_q f_{\text{prev}}^1 \quad \text{at } x=\alpha,$$

(5.7) $$T_{p-q}\beta u^2-\mu T_{p-q}u_x^2=T_{p-q}f_{\text{prev}}^2 \quad \text{at } x=\alpha,$$

(5.8) $$T_q u^2=T_q u^1 \quad \text{at } x=\alpha,$$

(5.9) $$T_{p-q}u^1=T_{p-q}u^2 \quad \text{at } x=\alpha,$$

(5.10) $$T_q\beta u^2+\lambda T_q u_x^2=T_q f_{\text{prev}}^2 \quad \text{at } x=1,$$

(5.11) $$T_{p-q}\beta u^1-\mu T_{p-q}u_x^1=T_{p-q}f_{\text{prev}}^1 \quad \text{at } x=-1,$$

(5.12) $$u_1^1=0 \quad \text{at } x=-1,$$

(5.13) $$u_1^2=0 \quad \text{at } x=1.$$

In the equations above β is the inverse of the timestep, $f_{\text{prev}}^1=f^1+\beta u_{\text{prev}}^1$ where u_{prev}^1 denotes the value of u_N^1 at the former level t^k, and f_{prev}^2 is defined similarly. Note that in the current case T_q and T_{p-q} reduce to the first and second row of the matrix T, respectively. The two equations (5.8) and (5.9) correspond exactly to the continuity condition (4.12).

Let us now apply to the problem (5.4)-(5.13) the iteration-by-subdomain method described in the previous section. At the $(n+1)$th step, the solutions $(u^1)^{n+1}$ and $(u^2)^{n+1}$ satisfy (5.4)-(5.7) and (5.10)-(5.13); in accordance with (4.20) and (4.25) the matching interface conditions (5.8) and (5.9) are dealt with as follows:

(5.14) $$T_q(u^2)^{n+1}=T_q(u^1)^n \quad \text{at } x=\alpha,$$

(5.15) $$T_{p-q}(u^1)^{n+1}=T_{p-q}(u^2)^n \quad \text{at } x=\alpha.$$

The main result we will prove in this section is that the above iterative procedure converges, i.e., $(u^1)^n\to u^1$ and $(u^2)^n\to u^2$ as $n\to\infty$. Furthermore, the convergence rate is independent of the polynomial degree N.

The proof is rather involved; it is convenient to carry it out using the characteristic form of the equations. For that, let us define the error functions (in characteristic form) at the $(n+1)$th iteration as follows:

(5.16) $$(e^i)^{n+1}:=T[(u^i)^{n+1}-u^i] \quad \text{for } i=1,2.$$

By comparing (5.4)-(5.13) with the equations satisfied by the iterates $(u^i)^{n+1}$, $i=1, 2$, it is readily seen that the following set of *error equations* hold:

Within Ω_1 (as usual, subindex denotes component):

(5.17.1) $$\beta(e^1)^{n+1}+\wedge(e^1)_x^{n+1}=0 \quad \text{at } x_j^1, \quad 1\leqq j\leqq N-1,$$

(5.17.2) $$(e_1^1)^{n+1}+(e_2^1)^{n+1}=0 \quad \text{at } x=-1,$$

(5.17.3) $$\beta(e_2^1)^{n+1}-\mu(e_2^1)_x^{n+1}=0 \quad \text{at } x=-1,$$

(5.17.4) $$\beta(e_1^1)^{n+1}+\lambda(e_1^1)_x^{n+1}=0 \quad \text{at } x=\alpha,$$

(5.17.5) $$(e_2^1)^{n+1}=(e_2^2)^n \quad \text{at } x=\alpha.$$

Within Ω_2:

(5.18.1) $$\beta(e^2)^{n+1}+\wedge(e^2)_x^{n+1}=0 \quad \text{at } x_j^2, \quad 1\leqq j\leqq N-1,$$

(5.18.2) $$(e_1^2)^{n+1}+(e_2^2)^{n+1}=0 \quad \text{at } x=1,$$

(5.18.3) $$\beta(e_1^2)^{n+1}+\lambda(e_1^2)_x^{n+1}=0 \quad \text{at } x=1,$$

(5.18.4) $$\beta(e_2^2)^{n+1}-\mu(e_2^2)_x^{n+1}=0 \quad \text{at } x=\alpha,$$

(5.18.5) $$(e_1^2)^{n+1}=(e_1^1)^n \quad \text{at } x=\alpha.$$

We start proving two important lemmas.

LEMMA 5.1. *Let $\eta>0$ and ε be two given real numbers, and consider the following "backward" collocation problem in the reference interval $[-1, 1]$. Find $u\in\mathbb{P}_N$ such that*

(5.19.1) $$u-\eta u_x=0 \quad \text{at } t_j, \quad j=1,\cdots, N,$$

(5.19.2) $$u=\varepsilon \quad \text{at } t_0=1$$

where, as usual, $t_j=\cos \pi_j/N, j=0,\cdots, N$ are the Chebyshev-Lobatto points in $[-1, 1]$. There exists a constant $\sigma_N(\eta)$ whose absolute value is strictly less than one such that

(5.20) $$u(-1)=\sigma_N(\eta)\varepsilon.$$

Proof. Let $T_k(x)$ denote the Chebyshev polynomial of degree k. (We recall that $T_k(x)=\cos k\theta$, where $\theta=\text{arcos } x$.) Since for $j=1,\cdots, N-1$, t_j are the roots of $T'_N(x)$, from (5.19.1) we deduce the following identity:

(5.21) $$u-\eta u_x=\tau(\phi_{N-1}+\phi_N) \quad \forall x\in[-1, 1]$$

where $\phi_{N-1}(x)=T'_N(x)$, $\phi_N(x)=xT'_N(x)$, and τ is a constant that can be determined using the inflow boundary condition (5.19.2). Following [1] we can state that

(5.22) $$\text{degree } (\phi_n)=n, \quad \text{parity of } \phi_n=\text{parity of } n, \quad (\hat{\phi}_n)_k\geqq 0$$

for $n=N-1$, N and $k=0,\cdots, n$. Here $(\hat{\phi}_n)_k$ are the Chebyshev coefficients of ϕ_n, i.e.,

(5.23) $$(\hat{\phi}_n)_k=\frac{2}{\pi c_k}\int_{-1}^{1}\phi_n(x)T_k(x)\frac{1}{\sqrt{1-x^2}}\,dx \quad \text{with } c_0=2 \text{ and } c_k=1 \text{ if } k\geqq 1.$$

Owing to (5.21), we can set

(5.24) $$u=\tau(\varphi_\eta+\psi_\eta)$$

where φ_η and ψ_η are the unique solutions of the following ordinary differential equations:

(5.25) $$\varphi_\eta\in\mathbb{P}_N: \varphi_\eta-\eta\varphi_{\eta,x}=\phi_{N-1},$$

(5.26) $$\psi_\eta\in\mathbb{P}_N: \psi_\eta-\eta\psi_{\eta,x}=\phi_N.$$

If p is any polynomial of $\mathbb{P}_N$ we recall that (e.g., [4, Chap. 2])

$$(5.27)\qquad p(x)=\sum_{k=0}^{N}\hat{p}_kT_k(x),\quad p_x(x)=\sum_{k=0}^{N-1}\hat{p}_k^{(1)}T_k(x)\quad \text{with } \hat{p}_k^{(1)}=\frac{2}{c_k}\sum_{\substack{m\geqq k+1\\ m\not\asymp k}} m\hat{p}_m$$

where the symbol $m\not\asymp k$ means that $|m-k|$ is odd. Taking into account (5.27), from (5.25) and (5.26), we obtain the following recurrence relations for the Chebyshev coefficients of φ_η and ψ_η:

$$(5.28)\qquad (\hat{\varphi}_\eta)_m=(\hat{\phi}_{N-1})_m+\eta\frac{2}{c_m}\sum_{\substack{k\geqq m+1\\ k\not\asymp m}} k(\hat{\varphi}_\eta)_k,\qquad m=N, N-1,\cdots,0,$$

$$(5.29)\qquad (\hat{\psi}_\eta)_m=(\hat{\phi}_{N})_m+\eta\frac{2}{c_m}\sum_{\substack{k\geqq m+1\\ k\not\asymp m}} k(\hat{\psi}_\eta)_k,\quad m=N, N-1,\cdots,0.$$

Owing to (5.22), we conclude that

$$(5.30)\qquad (\hat{\varphi}_\eta)_m\geqq 0,\quad (\hat{\psi}_\eta)_m\geqq 0\quad \text{for } m=0,\cdots,N\quad \text{for all } \eta>0.$$

moreover, noting that $T_k(1)=1$ for all k, from (5.24) we obtain

$$(5.31)\qquad \tau=\varepsilon\Big/\sum_{m=0}^{N}((\hat{\varphi}_\eta)_m+(\hat{\psi}_\eta)_m)\quad \text{whence sign } \tau=\text{sign } \varepsilon.$$

It follows that the solution to the collocation problem (5.19) has the following Chebyshev coefficients:

$$(5.32)\qquad \hat{u}_m=((\hat{\varphi}_\eta)_m+(\hat{\psi}_\eta)_m)\varepsilon\Big/\sum_{k=0}^{N}((\hat{\varphi}_\eta)_k+(\hat{\psi}_\eta)_k)$$

and therefore

$$(5.33)\qquad \text{sign } \hat{u}_m=\text{constant}=\text{sign } \varepsilon,\qquad m=0,\cdots,N.$$

Noting that $u(-1)=\sum_{m=0}^{N}\hat{u}_m(-1)^m$ as $T_m(-1)=(-1)^m$, we deduce (5.20) from (5.32) by setting

$$(5.34)\qquad \sigma_N(\eta):=\sum_{m=0}^{N}(-1)^m((\hat{\varphi}_\eta)_m+(\hat{\psi}_\eta)_m)\Big/\sum_{k=0}^{N}((\hat{\varphi}_\eta)_k+(\hat{\psi}_\eta)_k).$$

Finally, the property

$$(5.35)\qquad |\sigma_N(\eta)|<1\quad \forall\eta>0$$

follows from (5.30). □

LEMMA 5.2. *Let $\eta>0$ and ε be two given real numbers, and consider the following "forward" collocation problem in the reference interval* $[-1,1]$: *Find $v\in\mathbb{P}_N$ such that*

$$(5.36.1)\qquad v+\eta v_x=0\quad \text{at } t_j,\quad j=0,\cdots,N-1,$$

$$(5.36.2)\qquad v=\varepsilon\quad \text{at } t_N=-1$$

where the points t_j are defined as in Lemma 5.1. *Then*

$$(5.37)\qquad v(1)=\sigma_N(\eta)\varepsilon$$

where σ_N is the same as in (5.34).

Proof. First we note that

$$(5.38)\qquad v+\eta v_x=\rho(\phi_{N-1}-\phi_N)\quad \forall x\in[-1,1],\quad \rho=\frac{\varepsilon+\eta v_x(-1)}{2(-1)^{N-1}N^2}.$$

The constant ρ has been determined by the boundary condition (5.36.2), noting that $T'_k(1)=k^2$ and $T'_k(-1)=(-1)^{k+1}k^2$. Consider now the following "backward" problem associated with (5.36). Look for $w\in\mathbb{P}_N$ subject to

$$w-\eta w_x=0 \quad \text{at } t_j, \quad j=1,\cdots,N,$$
$$w=\varepsilon \quad \text{at } t_0=1.$$

We want to show that

$$v(-x)=w(x) \quad \forall x\in[-1,1]. \tag{5.39}$$

As a matter of fact, w verifies

$$w-\eta w_x=\rho^*(\phi_{N-1}+\phi_N) \quad \forall x\in[-1,1], \quad \rho^*=\frac{\varepsilon-\eta w_x(1)}{2N^2}. \tag{5.40}$$

Writing (5.40) at the point $-x$, we obtain

$$w(-x)-\eta w_x(-x)=\rho^*(1-x)T'_N(-x) \quad \forall x\in[-1,1]$$

whence, in view of (5.39),

$$v(x)+\eta v_x(x)=[(-1)^{N-1}\rho^*](1-x)T'_N(x). \tag{5.41}$$

We have used the property that $T'_k(-x)=(-1)^{k+1}T'_k(x)$, for all $x\in[-1,1]$ and the obvious relation: $w_x(-x)=-v_x(x)$. Since $(-1)^{N-1}\rho^*=\rho$ and $(1-x)T'_N(x)=\phi_{N-1}-\phi_N$, we conclude from (5.41) that v satisfies (5.38), whence v is the only solution of (5.36).

We can now apply the previous lemma to get $w(-1)=\sigma_N(\eta)\varepsilon$ and therefore (5.37) follows due to (5.39).

Furthermore, since the Chebyshev coefficients of w are given by (5.32), again using (5.39) we deduce that the Chebyshev coefficients of v are

$$\hat{v}_m=(-1)^m[(\hat{\varphi}_\eta)_m+(\hat{\psi}_\eta)_m]\varepsilon\Big/\sum_{k=0}^{N}((\hat{\varphi}_\eta)_k+(\hat{\psi}_\eta)_k). \tag{5.42}$$

In view of (5.20) and (5.37), the factor defined in (5.34) will be called the *outflow/inflow ratio.* □

We are now in the position to state the following result.

THEOREM 5.1. *If we define*

$$\lambda'=2\lambda/\beta(1+\alpha), \qquad \mu'=2\mu/\beta(1+\alpha) \tag{5.43}$$

the solution to the problem (5.17) *satisfies*

$$(e_1^1)^{n+1}(\alpha)=-\sigma_N(\lambda')\sigma_N(\mu')(e_2^2)^n(\alpha). \tag{5.44}$$

Proof. Let $t(x)=2(x+1)/(\alpha+1)-1$ be the affine transformation from $\bar{\Omega}_1=[-1,\alpha]$ to the reference interval $[-1,1]$. Let us define

$$u\in\mathbb{P}_N: u(t(x))=(e_2^1)^{n+1}(x) \quad \forall x\in\bar{\Omega}_1. \tag{5.45}$$

It is readily seen from (5.17.1), (5.17.3), and (5.17.5) that u is the solution to a backward collocation problem such as (5.19), provided we set $\eta=\mu'$ and $\varepsilon=(e_2^2)^n(\alpha)$. By (5.20) we therefore deduce

$$(e_2^1)^{n+1}(-1)=u(-1)=\sigma_N(\mu')(e_2^2)^n(\alpha). \tag{5.46}$$

Let us now define:

$$v\in\mathbb{P}_N: v(t(x))=(e_1^1)^{n+1}(x) \quad \forall x\in\bar{\Omega}_1. \tag{5.47}$$

Owing to (5.17.1), (5.17.4), and (5.17.2), it follows that v is the solution in $[-1,1]$ of the forward problem (5.36), provided we set $\eta=\lambda'$ and $\varepsilon=-(e_2^1)^{n+1}(-1)$. We can therefore apply the result (5.37), and owing to (5.47) and (5.46) we obtain

$$(e_1^1)^{n+1}(\alpha)=v(1)=\sigma_N(\lambda')\varepsilon=-\sigma_N(\lambda')\sigma_N(\mu')(e_2^2)^n(\alpha). \qquad \square$$

The following result is the counterpart of Theorem 5.1 for the subdomain Ω_2.

THEOREM 5.2. *Let us set*

$$\lambda''=2\lambda/\beta(1-\alpha) \quad \textit{and} \quad \mu''=2\mu/\beta(1-\alpha). \tag{5.48}$$

The solution to the problem (5.18) *satisfies*

$$(e_2^2)^{n+1}(\alpha)=-\sigma_N(\lambda'')\sigma_N(\mu'')(e_1^1)^n(\alpha). \tag{5.49}$$

Proof. Let $t(x)=2(x-\alpha)/(1-\alpha)-1$ be the affine mapping from $\bar{\Omega}_2=[\alpha,1]$ into $[-1,1]$, and define

$$v\in\mathbb{P}_N: v(t(x))=(e_1^2)^{n+1}(x) \quad \forall x\in\bar{\Omega}_2. \tag{5.50}$$

Owing to (5.18.1), (5.18.3), and (5.18.5) we deduce that in $[-1,1]$ v satisfies a problem such as (5.36) with η replaced by λ'' and ε by $(e_1^1)^n(\alpha)$. In view of (5.37) we therefore have

$$(e_1^2)^{n+1}(1)=\sigma_N(\lambda'')(e_1^1)^n(\alpha). \tag{5.51}$$

On the other hand, from (5.18.1), (5.18.2), and (5.18.4) it follows that the function

$$u\in\mathbb{P}_N: u(t(x))=(e_2^2)^{n+1}(x) \quad \forall x\in\bar{\Omega}_2 \tag{5.52}$$

satisfies on $[-1,1]$ a problem such as (5.19) provided η is replaced by μ'' and ε by $-(e_1^2)^{n+1}(1)$. Thus from (5.20) we obtain

$$(e_2^2)^{n+1}(\alpha)=-\sigma_N(\mu'')(e_1^2)^{n+1}(1).$$

Now (5.49) follows from (5.51). $\square$

Let us define the following sequence of *interface errors*:

$$E^n=[(e_1^1)^n(\alpha)]^2+[(e_2^1)^n(\alpha)]^2+[(e_1^2)^n(\alpha)]^2+[(e_2^2)^n(\alpha)]^2 \quad \text{for } n\geqq 1. \tag{5.53}$$

From the previous theorems we deduce the following convergence result.

THEOREM 5.3. *The interface error defined by* (5.53) *reduces at each interaction according to the law*

$$E^{n+1}\leqq\sigma_N^*(\lambda,\mu;\alpha)E^n, \qquad n\geqq 2 \tag{5.54}$$

where the reduction factor is defined as follows:

$$\sigma_N^*(\lambda,\mu;\alpha):=\max\{\sigma_N^2(\lambda')\sigma_N^2(\mu'),\sigma_N^2(\lambda'')\sigma_N^2(\mu'')\}<1. \tag{5.55}$$

Proof. Owing to (5.17.5), (5.18.5) and to the Theorems 5.3 and 5.4, we obtain that the interface errors propagate according to the following relation:

$$E^{n+1}=\{[(e_2^1)^n]^2+[(e_2^2)^n]^2\}\sigma_N^2(\lambda')\sigma_N^2(\mu')+\{[(e_1^1)^n]^2+[(e_1^2)^n]^2\}\sigma_N^2(\lambda'')\sigma_N^2(\mu'') \quad \text{for } n\geqq 2. \tag{5.56}$$

The inequality (5.54) follows easily. Note that $\sigma_N^*(\lambda,\mu;\alpha)<1$ for all positive λ and μ as a consequence of (5.35). $\square$

Remark 5.1 (behaviour of the error reduction factor). The behaviour of $\sigma_N^*(\lambda,\mu;\alpha)$ (and, by consequence, that of the interface error sequence (5.53)) is driven by the behaviour of the outflow/inflow ratio defined in (5.34).

From Tables 5.1 and 5.2 we see that for all values of N and η, $\sigma_N(\eta) > 0$. Moreover, for any fixed value of η, $\sigma_N(\eta)$ is uniformly bounded from above by a constant strictly less than one as N increases (even better, the value of $\sigma_N(\eta)$ is substantially independent of N). This property is very important as it ensures that the error reduction factor does not approach one as N tends to infinity, therefore yielding a convergence rate for our iterative procedure that is (practically) independent of N. In Table 5.2 we report the limit of $\sigma_N(\eta)$ as $N \to \infty$, for several values of η. It is apparent that

$$\lim_{N \nearrow \infty} \sigma_N(\eta) = e^{-2/\eta} \quad \forall \eta > 0. \tag{5.57}$$

Thus, the limit is precisely the outflow/inflow ratio of the differential case, i.e., the value $v(-1)/v(1)$, where v is the exact solution to the ordinary differential equation $v - \eta v_x = 0$.

On the other hand, for fixed N, $\sigma_N(\eta)$ behaves like a monotonically increasing function of η (if N and/or η are not too small). In the current application, η takes the values of λ', λ'', μ', and μ'', which are all proportional to the timestep $\Delta t = \beta^{-1}$ (see (5.43) and (5.48)). Therefore, for large N we have approximately

$$\sigma_N^*(\lambda, \mu; \alpha) \simeq \exp\left\{-\frac{1}{\Delta t} 2\alpha^*(\lambda + \mu)/\lambda\mu\right\}, \qquad \alpha^* = \min(1+\alpha, 1-\alpha). \tag{5.58}$$

We recall that λ and $-\mu$ are the eigenvalues of the transport matrix A (see (5.1)), while $x = \alpha$ is the abscissa of the interface between the two subdomains Ω_1 and Ω_2. Note that α^* is the minimum measure of the subdomains. □

TABLE 5.1
The value of the outflow/inflow ratio $\sigma_N(\eta)$ for several values of η and N.

η \ N	4	8	16	32
0.01	0.10195409	4.5329018E−2	2.5022730E−4	9.4749803E−11
0.05	7.6327433E−2	1.8829854E−2	1.9753051E−4	1.8740999E−11
0.1	5.8272078E−2	6.0076584E−3	1.7829169E−6	2.0611536E−9
0.5	2.8037383E−2	1.8321630E−2	1.8315638E−2	1.8315638E−2
1	0.13687601	0.13533534	0.13533528	0.13533528
5	0.67032260	0.67032005	0.67032005	0.67032005
10	0.81873085	0.81873075	0.81873075	0.81873075
100	0.98019867	0.98019867	0.98019867	0.98019867

TABLE 5.2
The value of $\sigma_\infty(\eta) := \lim_{N\to\infty} \sigma_N(\eta)$ for several values of η.

η	$\sigma_\infty(\eta)$
0.01	0
0.05	4.24835425E−18
0.1	2.0611536E−9
0.5	1.8315638E−2
1	0.13533528
5	0.67032005
10	0.81873075
100	0.98019867

From the previous theorem we have that

$$\lim_{n\to\infty} E^n = 0. \tag{5.59}$$

To conclude that our iteration-by-subdomain procedure converges it is enough to prove that the error functions $(e^i)^n(x)$, $i=1,2$, defined in (5.16) attain their maximum values at the interface point $x=\alpha$. This is stated by the following theorem.

THEOREM 5.4. *With the notation* (5.16) *and* (5.23), *we have*

$$\begin{aligned}&\max_{-1\leqq x\leqq \alpha}\{[(e_1^1)^n(x)]^2+[(e_2^1)^n(x)]^2\}\\ &\quad+\max_{\alpha\leqq x\leqq 1}\{[(e_1^2)^n(x)]^2+[(e_2^2)^n(x)]^2\}\leqq E^n \quad \forall n\geqq 1.\end{aligned} \tag{5.60}$$

Proof. From the proof of Theorem 5.1 we easily deduce that for all $x\in[-1,\alpha]$

$$(e_2^1)^{n+1}(x)=\sum_{m=0}^{N}\hat{u}_m T_m(t(x))=\frac{(e_2^2)^n(\alpha)}{\sum_{k=0}^{N}[(\hat{\varphi}_{\mu'})_k+(\hat{\psi}_{\mu'})_k]}\sum_{m=0}^{N}[(\hat{\varphi}_{\mu'})_m+(\hat{\psi}_{\mu'})_m]T_m(t(x)) \tag{5.61}$$

and

$$\begin{aligned}(e_1^1)^{n+1}(x)&=\sum_{m=0}^{N}\hat{v}_m T_m(t(x))\\ &=-\sigma_N(\mu')\frac{(e_2^2)^n(\alpha)}{\sum_{k=0}^{N}[(\hat{\varphi}_{\lambda'})_k+(\hat{\psi}_{\lambda'})_k]}\sum_{m=0}^{N}[(\hat{\varphi}_{\lambda'})_m+(\hat{\psi}_{\lambda'})_m]T_m(t(x)).\end{aligned} \tag{5.62}$$

Owing to (5.30) and the fact that $|T_m(\xi)|\leqq 1$, for all $\xi\in[-1,1]$, we deduce from the previous equalities and from (5.17.5) that for all $x\in[-1,\alpha]$

$$|(e_2^1)^{n+1}(x)|\leqq|(e_2^1)^{n+1}(\alpha)|,\qquad |(e_1^1)^{n+1}(x)|\leqq|\sigma_N(\mu')||(e_2^1)^{n+1}(\alpha)|. \tag{5.63}$$

In a similar way, using Theorem 5.2 we prove that for all $x\in[\alpha,1]$

$$|(e_1^2)^{n+1}(x)|\leqq|(e_1^2)^{n+1}(\alpha)|,\qquad |(e_2^2)^{n+1}(x)|\leqq|\sigma_N(\lambda'')||(e_1^2)^{n+1}(\alpha)|. \tag{5.64}$$

Inequality (5.60) follows easily from (5.61)-(5.64). □

We can now state our final convergence result.

COROLLARY 5.1. *The iteration-by-subdomain procedure applied to the multidomain Chebyshev collocation problem* (5.4)-(5.13) *converges as* $n\to\infty$.

Proof. The result follows from (5.60) and (5.59). Actually, since T is nonsingular, the convergence of the sequence $(e^i)^n$ implies that of $(u^i)^n-u^i$ owing to (5.16). □

Remark 5.2. The same kind of result holds if the spectral Legendre collocation method is used instead of the Chebyshev. □

Remark 5.3 (convergence for decomposition with several subdomains). For a decomposition of $[-1,1]$ using more than two subdomains we can essentially carry out the same type of convergence proof. Putting aside the technical difficulties, it is not hard to see that in this case the error reduction factor at each interface behaves as does (5.58), where now α^* is the measure of the smallest subdomain of the decomposition.

If, for instance, the computational domain $[-1,1]$ is partitioned into M subdomains of equal length, the error reduction factor behaves as does

$$\exp\left\{-\frac{1}{\Delta t}\frac{4}{M}\frac{\lambda+\mu}{\lambda\mu}\right\}, \tag{5.65}$$

hence the convergence rate slows down as far as M increases.

Such a behaviour is somehow physiological for the method at hand. As a matter of fact, breaking the computational domain in M pieces and iterating on the subdomains yields that the time for the wave propagation throughout the whole domain is finite.

A remedy can be the use of a multiple scale of subdomain partitions, starting with a coarse subdivision in two subdomains and going progressively up to the desired finest partition in M subdomains, then back to two subdomains and so forth. At any level, the same polynomial degree N should be used within each subdomain. The strategy here described can be cast in a precise mathematical framework. Actually, it is precisely readable as a multigrid method for the capacitance matrix that we are going to introduce in next section. Since the capacitance matrix affects the interface unknowns only, reducing the number of subdomains amounts to reducing the number of interfaces, and, by consequence, the size of the capacitance matrix. A precise mathematical investigation on this multilevel method is in progress. □

Remark 5.4 (computational complexity of multidomain versus single-domain approximations). Implicit finite differences for advancing in time the multidomain spatial discretization yield at each step a block linear system. Each block can be expressed in terms of the Chebyshev pseudospectral differentiation matrix D, whose multiplication by a vector $\mathbf{u}$ of gridvalues at the Chebyshev points produces the gridvalues at the same points of the polynomial of degree N interpolating $\mathbf{u}$. The condition number of the system grows quadratically in terms of N (e.g., [4, § 4.2]). Unfortunately, the lack of effective preconditioners for the matrix D makes the use of direct methods mandatory for the solution of our system. Since the domain decomposition approach yields a sequence of systems of reduced size, its computational complexity is lower than that of the single domain approach. For instance in one dimension using polynomials of degree N within each subdomain would give an order of $M^2N^3/3$ operations (assuming a number of iterations proportional to M, as predicted), versus an order of $N^3M^3/3$ operations using single domain approach with the same number $N \cdot M$ of gridpoints. The gap becomes larger in higher space dimensions, and a similar conclusion holds, of course, for the memory requirement. Last but not least, domain decomposition can be mandatory to face computational domains that are not amenable to a Cartesian geometry by a simple mapping. □

6. Capacitance matrix interpretation. We now construct the capacitance (or Schur complement) matrix associated with the multidomain problem (5.4)-(5.13). First, we rewrite the problem in terms of the characteristic variables

$$z^1 = Tu^1, \qquad z^2 = Tu^2. \tag{6.1}$$

If we define $h^1 = Tf^1_{\text{prev}}$ and $h^2 = Tf^2_{\text{prev}}$, from (5.4)-(5.13) we obtain

$$\beta z^1_1 + \lambda z^1_{1,x} = h^1_1 \quad \text{at } x^1_j, \quad j = 0, \cdots, N-1, \tag{6.2.1}$$

$$\beta z^1_2 - \mu z^1_{2,x} = h^1_2 \quad \text{at } x^1_j, \quad j = 1, \cdots, N, \tag{6.2.2}$$

$$z^1_1 + z^1_2 = 0 \quad \text{at } x^1_N = -1, \tag{6.2.3}$$

$$z^1_2 = z^2_2 \quad \text{at } x = \alpha, \tag{6.2.4}$$

$$z^2_1 = z^1_1 \quad \text{at } x = \alpha, \tag{6.2.5}$$

$$z^2_1 + z^2_2 = 0 \quad \text{at } x^2_0 = 1, \tag{6.2.6}$$

$$\beta z^2_1 + \lambda z^2_{1,x} = h^2_1 \quad \text{at } x^2_j, \quad j = 0, \cdots, N-1, \tag{6.2.7}$$

$$\beta z^2_2 - \mu z^2_{2,x} = h^2_2 \quad \text{at } x^2_j, \quad j = 1, \cdots, N. \tag{6.2.8}$$

Note that (6.2.4) and (6.2.5) are equivalent to (5.9) and (5.8), respectively. The capacitance (or Schur complement) system is the one that allows the determination of the values of the two characteristic variables z_1^1 and z_2^1 (and henceforth, those of z_1^2 and z_2^2 owing to the continuity relations (6.2.4) and (6.2.5)) at the interface $x=\alpha$.

Consider the polynomial solutions to the two following multidomain collocation problems:

(6.3)
$$\begin{aligned}
&\beta U_1^1+\lambda U_{1,x}^1=0 \quad \text{at } x_j^1, \quad j=0,\cdots,N-1,\\
&\beta U_2^1-\mu U_{2,x}^1=0 \quad \text{at } x_j^1, \quad j=1,\cdots,N,\\
&U_1^1+U_2^1=0 \quad \text{at } x_N^1=-1,\\
&U_2^1=1 \quad \text{at } x=\alpha,
\end{aligned}$$

(6.4)
$$\begin{aligned}
&\beta U_1^2+\lambda U_{1,x}^2=0 \quad \text{at } x_j^2, \quad j=0,\cdots,N-1,\\
&\beta U_2^2-\mu U_{2,x}^2=0 \quad \text{at } x_j^2, \quad j=1,\cdots,N,\\
&U_1^2+U_2^2=0 \quad \text{at } x_0^2=1,\\
&U_1^2=1 \quad \text{at } x=\alpha.
\end{aligned}$$

Furthermore, let us consider the two polynomial solutions to the multidomain collocation problems:

(6.5)
$$\begin{aligned}
&\beta V_1^1+\lambda V_{1,x}^1=h_1^1 \quad \text{at } x_j^1, \quad j=0,\cdots,N-1,\\
&\beta V_2^1-\mu V_{2,x}^1=h_2^1 \quad \text{at } x_j^1, \quad j=1,\cdots,N,\\
&V_1^1+V_2^1=0 \quad \text{at } x_N=-1,\\
&V_2^1=0 \quad \text{at } x=\alpha,
\end{aligned}$$

(6.6)
$$\begin{aligned}
&\beta V_1^2+\lambda V_{1,x}^2=h_1^2 \quad \text{at } x_j^2, \quad j=0,\cdots,N-1,\\
&\beta V_2^2-\mu V_{2,x}^2=h_2^2 \quad \text{at } x_j^2, \quad j=1,\cdots,N,\\
&V_1^2+V_2^2=0 \quad \text{at } x_0^2=1,\\
&V_1^2=0 \quad \text{at } x=\alpha.
\end{aligned}$$

Let us denote by ξ the (unknown) value of z_1^1 $(=z_1^2)$ at $x=\alpha$, and by η the (unknown) value of z_2^1 $(=z_2^2)$ at $x=\alpha$. The following relationships hold between the solution to the problem (6.2) and those of the auxiliary problems (6.3)-(6.6):

(6.7) $$z^1=V^1+\eta U^1 \quad \text{in } \Omega_1,$$

(6.8) $$z^2=V^2+\xi U^2 \quad \text{in } \Omega_2.$$

The values of ξ and η are therefore determined by the following two equations that arise from (6.2.4) and (6.2.5):

(6.9) $$V_1^1(\alpha)+\eta U_1^1(\alpha)=V_1^2(\alpha)+\xi U_1^2(\alpha),$$

(6.10) $$V_2^1(\alpha)+\eta U_2^1(\alpha)=V_2^2(\alpha)+\xi U_2^2(\alpha).$$

Using the last equations of (6.3)-(6.6) we obtain the 2×2 system:

(6.11) $$\begin{pmatrix} 1 & -U_1^1(\alpha)\\ -U_2^2(\alpha) & 1\end{pmatrix}\begin{pmatrix}\xi\\ \eta\end{pmatrix}=\begin{pmatrix}V_1^1(\alpha)\\ V_2^2(\alpha)\end{pmatrix}.$$

The matrix in (6.11) is the *capacitance matrix* (it is also called influence or Schur complement matrix): hereafter it will be denoted by S.

To give a precise form to its entries, we note that by comparison of (5.17) with (6.3) U^1 and $(e^1)^{n+1}$ satisfy the same set of equations in Ω_1. The only difference regards the value attained at $x=\alpha$ by the second component of the solution (precisely, $U_2^1(\alpha)=1$ and $(e_2^1)^{n+1}(\alpha)=(e_2^2)^n(\alpha)$). Owing to (5.44), we can therefore conclude that

$$-U_1^1(\alpha)=\sigma_N(\lambda')\sigma_N(\mu') \tag{6.12}$$

with λ' and μ' given in (5.43).

Exploiting a similar analogy between U^2 and the solution of problem (5.18) we easily obtain from (5.49) that

$$-U_2^2(\alpha)=\sigma_N(\lambda'')\sigma_N(\mu'') \tag{6.13}$$

where λ'' and μ'' are defined in (5.48). Then

$$S=\begin{pmatrix} 1 & \sigma_N(\lambda')\sigma_N(\mu') \\ \sigma_N(\lambda'')\sigma_n(\mu'') & 1 \end{pmatrix}. \tag{6.14}$$

THEOREM 6.1. *The capacitance matrix is positive definite. Moreover, if $\alpha=0$ (i.e., Ω_1 and Ω_2 have the same measure), then S is symmetric and its eigenvalues are*

$$\lambda_{1,2}=1\pm\sigma_N(\lambda')\sigma_N(\mu'). \tag{6.15}$$

Proof. S is positive definite since $|\sigma_N(\eta)|<1$ for any positive η. The eigenvalues of S are

$$\lambda_{1,2}=1\pm\sqrt{\sigma_N(\lambda')\sigma_N(\mu')\sigma_N(\lambda'')\sigma_N(\mu'')}. \tag{6.16}$$

If meas $\Omega_1=$ meas Ω_2 (in our example, this means that $\alpha=0$), we have $\lambda'=\lambda''(=2\lambda/\beta)$ and $\mu'=\mu''(=2\mu/\beta)$, whence

$$\sigma_N(\lambda')=\sigma_N(\lambda''), \qquad \sigma_N(\mu')=\sigma_N(\mu'').$$

Thus S is symmetric, and its eigenvalues are given by (6.15) owing to (6.16). $\square$

We conclude this section by establishing an important relationship between the matrix S and the iteration-by-subdomain method we described in the previous section.

Let us consider the spectral multidomain problem (5.4)-(5.13), and denote by $\Xi=(\xi,\eta)^t$ the unknown vector of the characteristic solution at the interface point $x=\alpha$, i.e.,

$$\xi=z_1^1(\alpha)=z_1^2(\alpha), \quad \eta=z_2^1(\alpha)=z_2^2(\alpha). \tag{6.17}$$

By a straightforward calculation it is not hard to see that one step of our iteration-by-subdomain procedure amounts to applying one step of the Richardson iterative method to the capacitance system (6.11). Precisely, going from the step n to the step $n+1$ produces a change on the interface variables Ξ that is ruled by the following relationship:

$$\Xi^{n+1}=\Xi^n+(R-S\Xi^n). \tag{6.18}$$

Here we have denoted by R the right-hand side of (6.11).

Using in (6.18) an acceleration parameter ω a priori different than one, i.e., considering the following sequence

$$\Xi^{n+1}=\Xi^n+\omega(R-S\Xi^n), \qquad \omega>0, \tag{6.19}$$

amounts to replacing the interface matching conditions (5.14)-(5.15) by the new ones:

$$T_q(u^2)^{n+1}=\omega T_q(u^1)^n+(1-\omega)T_q(u^2)^n \quad \text{at } x=\alpha, \tag{6.20}$$

$$T_{p-q}(u^1)^{n+1}=\omega T_{p-q}(u^2)^n+(1-\omega)T_{p-q}(u^1)^n \quad \text{at } x=\alpha. \tag{6.21}$$

The other statements of the iterative algorithm hold unchanged. In (6.20)-(6.21) ω plays the role of a relaxation factor. In view of (6.19), it is well known that whenever S is symmetric and positive definite, the best choice of ω is (see [15])

$$\omega_{\text{opt}} = 2/(\lambda_{\min} + \lambda_{\max})$$

where $\lambda_{\min}$ and $\lambda_{\max}$ are the minimum and maximum eigenvalues of S. Thus if meas (Ω_1) = meas (Ω_2), in view of Theorem 5.5 we conclude that $\omega_{\text{opt}} = 1$.

7. Numerical results. We show the potential capability of our domain decomposition method to yield effective solutions of hyperbolic systems by presenting some numerical results for the simple problem considered in § 5. However, we allow the nonhomogeneous boundary values in (5.3).

The time discretization method considered is the second-order Crank-Nicolson scheme. We considered the case of two different sets of initial and boundary data, so that the corresponding exact solutions are

$$\text{First test function:}\quad u^1(x,t) = e^t \cos \alpha x, \quad u^2(x,t) = e^t \sin \alpha x,$$

$$\text{Second test function:}\quad u^1(x,t) = e^t \operatorname{arctg} \beta(x+0.5), \quad u^2(x,t) = e^t \operatorname{arctg} \beta(x-0.5)$$

where α and β are some given parameters.

In all cases, we display the results obtained at the time $t=1$ for several values of the timestep Δt, the polynomial degree N used inside each subdomain, and the number M of subdomains. In Tables 7.1 and 7.2 we report the maximum norm of the relative error between the approximate and exact solutions at the time $t=1$ using $\Delta t = 0.01$. The tables refer to the first and second test functions, respectively, for single domain approximations and for approximations with two subdomains of equal length.

As we can see the better accuracy we obtain using two subdomains, rather than simply one, is more relevant if the expected solutions exhibits important oscillations (and/or variations) within the computational domain. Furthermore, since the matrices associated with the spectral collocation method are full, the use of several subdomains,

TABLE 7.1
First test function, one and two subdomain approximations.

N \ α	One domain			N \ α	Two subdomains		
	1	4	10		1	4	10
4	1.361E−4	2.429E−2	0.184	4	1.599E−5	1.059E−2	0.747
8	1.472E−7	8.134E−3	3.741E−1	8	1.436E−7	1.311E−5	7.266E−2
20	1.472E−7	1.546E−7	3.718E−4	12	1.363E−7	1.548E−7	1.552E−3

TABLE 7.2
Second test function, one and two subdomain approximations.

N \ β	One domain			N \ β	Two subdomains		
	1	4	10		1	4	10
4	4.767E−4	1.288E−2	2.812E−2	4	1.937E−4	6.119E−3	3.668E−2
8	1.094E−4	5.079E−3	2.345E−2	8	7.377E−7	1.024E−3	1.893E−2
20	2.572E−7	2.860E−4	1.005E−2	12	2.577E−7	1.943E−4	9.231E−3

TABLE 7.3
Number of iterations for the second test function when $\beta = 10$ and $\Delta t = 0.1$ (within the brackets are the values for the case $\Delta t = 0.01$).

N \ M	2	4	8	12
4	2 (2)	5 (4)	8 (5)	12 (5)
12	2 (2)	4 (4)	8 (5)	12 (5)
20	2 (2)	4 (4)	8 (4)	12 (4)

TABLE 7.4
Number of iterations for the first test function where $\alpha = 1$ and $N = 16$.

Δt \ M	2	4	8	12	16
0.1	2	4	8	12	16
0.02	2	4	4	4	5
0.01	2	4	4	4	4

accompanied with a sound tuning of the polynominal degree within each subdomain, allows the reduction of the overall complexity of the numerical problem.

We now report the results obtained by the iterative procedure presented in § 3 for the resolution of the multidomain problem.

In Tables 7.3 and 7.4 we report the number of iterations that are needed to damp the initial error by a factor of 10^{-5}. As usual M denotes the number of subdomains and N the polynomial degree of the discrete solution inside each subdomain.

Table 7.3 refers to the second test function when $\beta = 10$: it shows that the number of iterations is independent of N, while it grows at most linearly with M when Δt is "large." According to the convergence theory of § 5, the rate of convergence improves as the timestep Δt decreases. Finally, in the Table 7.4 we present the results for the first test function when $\alpha = 1$, obtained for $N = 16$ and several values of M and Δt.

Acknowledgments. I thank M. Salas, C. L. Streett, and T. A. Zang for many valuable discussions concerning this research.

REFERENCES

[1] C. CANUTO, *Spectral methods and a maximum principle*, Math. Comp., 51 (1988), pp. 615-630.
[2] T. F. CHAN, R. GLOWINSKI, J. PERIAUX, AND O. WIDLUND, EDS., *Domain Decomposition Methods for* P.D.E.*s*, II, Society for Industrial and Applied Mathematics, Philadelphia PA, 1988.
[3] L. CAMBIER, W. GHAZZI, J. P. VEUILLOT, AND H. VIVIAND, *Une approche par domaines pour le calcul d'ecoulements compressibles*, in Computer Methods in Applied Sciences and Engineering, V, R. Glowinski and J. L. Lions, eds., North-Holland, Amsterdam, 1982, pp. 423-446.
[4] C. CANUTO, M. Y. HUSSAINI, A. QUARTERONI, AND T. A. ZANG, *Spectral Methods in Fluid Dynamics*, Springer-Verlag, Berlin, Heidelberg, New York, 1988.
[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] D. GOTTLIEB, L. LUSTMAN, AND E. TADMOR, *Stability analysis of spectral methods for hyperbolic initial-boundary value systems*, in Methodes Spectrales, R. Teman, ed., Eyrolles, Paris, France, 1988, pp. 63-92.

[7] D. GOTTLIEB, L. LUSTMAN, AND E. TADMOR, *Convergence of spectral methods for hyperbolic initial-boundary value systems*, in Methodes Spectrales, R. Teman, ed., Eyrolles, Paris, France, 1988, pp. 51–62.

[8] D. GOTTLIEB, M. GUNZBURGER, AND E. TURKEL, *On numerical boundary treatment for hyperbolic systems*, SIAM J. Numer. Anal., 19 (1982), pp. 671–697.

[9] D. A. KOPRIVA, *A spectral multidomain method for the solution of hyperbolic systems*, Appl. Numer. Math., 2 (1986), pp. 221–241.

[10] P. D. LAX, *Hyperbolic Systems of Conservation Laws and the Mathematical Theory of Shock Waves*, CBMS-NSF Regional Conference Series in Applied Mathematics 11, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1973.

[11] M. MACARAEG AND C. L. STREETT, *An analysis of artificial viscosity effects on reacting flows using a spectral multi-domain technique*, in Computational Fluid Dynamics, G. de Vahl Davis and C. Fletcher, eds., North-Holland, Amsterdam, 1988, pp. 503–514.

[12] ———, *A spectral multi-domain technique applied to high-speed chemically reacting flows*, in Domain Decomposition Methods for P.D.E.s, T. F. Chan, R. Glowinski, J. Periaux, and O. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988.

[13] J. SMOLLER, *Shock Waves and Reaction-Diffusion Equations*, Springer-Verlag, Berlin, New York, 1983.

[14] R. TEMAM, ED., *Methodes Spectrales*, Collections de la Direction des Etudes et Recherches d'Electricité de France, n.68, Eyrolles, Paris, France, 1988.

[15] D. M. YOUNG, *Iterative Solution of Large Linear Systems*, Academic Press, New York, 1971.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 6, pp. 1053-1072, November 1990

MONTE CARLO SIMULATION OF LINEAR TWO-PHASE FLOW IN HETEROGENEOUS MEDIA*

F. MA† AND M. S. WEI‡

Abstract. In this paper an algorithmic formulation is given for the synthesis of linear two-phase systems with inherent variability. It is assumed that uncertainties in the flow system arise due to the heterogeneities of the porous medium. Methods of stochastic mechanics are adopted to offer a novel approach, whereby the absolute permeability is taken as a spatial stochastic process with a log-normal distribution at each point of the flow domain. A correlation structure is built into the model so as to achieve a realistic representation. In addition, a constraint on the invariance of the harmonic means of the random permeabilities is attained by way of topological scaling. A computer-synthesized waterflooding experiment is conducted to explore the effect of heterogeneity on output variables such as the oil pressure head and cumulative recovery. It is observed that output uncertainties can become significant even when the heterogeneities are small. By employing a least-squares procedure, the sensitivity of relative permeabilities to induced stochastic variations is also qualitatively investigated.

Key words. Monte Carlo simulation, linear two-phase flow, random processes, statistical correlation

AMS(MOS) subject classifications. 65C05, 65U05

1. Introduction.

It has long been recognized that the innate variability of a geophysical system is a complex phenomenological process that can only be described statistically. The existence of spatial randomness in various structural parameters of geologic formations has been admitted (Muskat (1949) and Anderson (1979)), but the extent and significance of such uncertainty are always in dispute. As a result, this awareness has tended to urge overdesign and conservatism in estimates of system performance rather than an organized study of the random effects. That, coincidentally, is also a reflection of the lack of a general formulation for the construction of stochastic mathematical models with inherent colored random variations. The heavy machinery of Itô calculus and the diffusional approach of Kolmogorov are both developed for white noise (Caughey (1971) and Ludwig (1975)), which often serves as the background excitation of many electronic and mechanical systems. The study of stochastic systems with colored, or nonwhite, parameters is far less extensive. Analytical techniques for colored noise are both scant and restrictive, and are usually applicable to models with either small input uncertainties or unbounded extent (Willems (1975) and Contreras (1980)). These limitations render most analytical methods inappropriate for the analysis of real-life systems. Simulation is the common procedure to be employed on colored stochastic systems.

Monte Carlo techniques can be easily applied to systems with large input uncertainties or defined over bounded domains. These techniques involve the repetitive evaluations of various system properties, and as such there is always the question of computational feasibility. The advent of modern computers, however, has now made the brute-force computational study of many stochastic problems acceptable. In fact, the design and survivability analysis of many important military and civilian systems

* Received by the editors December 21, 1988; accepted for publication August 28, 1989. This material is based in part upon work supported by National Science Foundation grant MSM-8657619. The government has certain rights in this material. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. This research has also been supported by the Standard Oil Company.

† Department of Mechanical Engineering, University of California, Berkeley, California 94720.

‡ Center for Naval Analyses, 4401 Ford Avenue, Alexandria, Virginia 22302.

are currently performed by Monte Carlo methods (Collins and Hudson (1981) and Rowan et al. (1974)). Coupled with the effective utilization of parallel processing, Monte Carlo analysis of stochastic problems will become increasingly attractive in the next decade as computing costs decrease. At the present time, one important trend of research in stochastic systems is the design of realistic models that are easily amenable to computational procedures.

This paper sets out to explore the synthesis of linear transient two-phase systems with inherent variability. Linear two-phase flow means the one-dimensional flow of two phases. It is assumed that uncertainties in the flow system arise due to the heterogeneities of the porous solid. The statistical variations within the finite flow field are represented by a spatial stochastic process. This leads to a stochastic differential equation, and subsequently a stochastic finite-difference scheme, for the flow. An algorithm is constructed to effectively capture the spatial correlation structure of the flow field. Monte Carlo techniques are employed for the evaluation of various system parameters, and one basic objective is to determine the effect of input uncertainty on the output. Monte Carlo simulation, as used in the title, refers to the solution of the stochastic differential equation by repetitive solution of deterministic realizations of this equation. It is decided that emphasis should be placed on algorithmic and conceptual developments. For this reason, only one-dimensional simulations will be pursued. As is usual, this paper builds upon several earlier studies (Warren and Price (1961), Smith and Freeze (1979), Smith and Brown (1982), and Ma and Wei (1985)). However, it is believed that the methodology reported herein has not previously been employed in earlier work. The organization of the paper is as follows. In § 2, the equations governing linear two-phase flow are examined. The equations are then recast in alternative forms for the purpose of numerical solution. The need for a stochastic model is addressed in § 3. The construction of the random flow field is expounded in §§ 4 and 5. Important observations on linear flow in random fields are highlighted in §§ 6 and 7. A summary of findings is given in § 8. Techniques expounded in this paper may be readily modified and applied to other areas of science and engineering. The reliability-based design of two-phase systems and other practical applications of the results will appear elsewhere.

2. Problem formulation. The equations governing transient multiphase flow through porous media are well known (Bear (1972) and Greenkorn (1983)). In the particular case of linear waterflooding in a horizontal direction, the evolution of the two-phase system composed of water and oil is described by the following analytical formulation. The conservation of mass for immiscible and incompressible phases can be expressed by the continuity equations

$$\phi\frac{\partial S_w}{\partial t}+\frac{\partial q_w}{\partial x}=0, \tag{1}$$

$$\phi\frac{\partial S_o}{\partial t}+\frac{\partial q_o}{\partial x}=0, \tag{2}$$

where S_w and S_o are water and oil saturations, q_w and q_o are water and oil flow rates per unit cross-sectional area of the core, ϕ is the porosity of the core, and x and t are dimensional space and time variables, respectively. When Darcy's law is applied to each phase to express the conservation of momentum, two additional equations arise

$$q_w=-\frac{kk_{rw}}{\mu_w}\frac{\partial P_w}{\partial x}, \tag{3}$$

$$q_o = -\frac{kk_{ro}}{\mu_o}\frac{\partial P_o}{\partial x}. \tag{4}$$

In the above, k refers to the absolute permeability of the core, and μ_w, P_w, k_{rw}, μ_o, P_o, k_{ro} are the viscosity, pressure, and relative permeability of the respective phase. It is assumed that k_{rw}, k_{ro} are functions of their saturations. However, by definition,

$$S_w + S_o = 1, \tag{5}$$

so that we need only select one saturation, say S_w, to be the sole variable on which k_{rw}, k_{ro} depend. Assuming throughout that water is the wetting phase, the pressures P_w, P_o in the two phases are related to each other through the capillary pressure P_c in such a way that

$$P_c = P_o - P_w. \tag{6}$$

The capillary pressure P_c is also a function of S_w. Let S_{wi} and S_{or} denote, respectively, the irreducible water saturation and residual oil saturation, both being constants of the system. Define the normalized water saturation S by

$$S(x,t) = \frac{S_w(x,t) - S_{wi}}{1 - S_{wi} - S_{or}}, \tag{7}$$

and assume the familiar parameterizations (Wyllie (1962) and Bear (1972))

$$k_{ro} = p_3(1-S)^{p_1} + p_4(1-S)^{p_2}, \tag{8}$$

$$k_{rw} = p_7 S^{p_5} + p_8 S^{p_6}, \tag{9}$$

$$P_c = -p_9 \ln(S). \tag{10}$$

The arbitrary constants p_i, $1 \leqq i \leqq 9$, are positive and satisfy the requirements that

$$p_3 + p_4 = k_{ro}^*, \tag{11}$$

$$p_7 + p_8 = k_{rw}^*, \tag{12}$$

where k_{ro}^* and k_{rw}^* are given values corresponding to the endpoints of the relative permeability curves, which can be measured in the laboratory. Due to constraints (11) and (12), it is clear that there are only seven free parameters involved in the parameterizations.

The set of four coupled partial differential equations (1)–(4), presented above, provides a fairly sophisticated analytical description of linear two-phase flow with capillarity. A subsequent task, then, is to attempt to solve this system of partial differential equations. However, it has been demonstrated, by a series of Lie–Backlund transformations, that the system of equations is not exactly solvable unless the capillary pressure assumes a specific functional form, or is zero (Douglas, Blair, and Wagner (1958), Fokas and Yortsos (1982), and Yortsos and Fokas (1983)). With the capillary pressure P_c given by (10), it becomes necessary to turn to numerical techniques for solution. To this end, assume that the water injection rate per unit cross-sectional area is constant and is equal to Q. Although no material simplification of the problem is effected by using a constant rate, it is convenient to do so, and throughout the remainder of the discussion this will be assumed. It is obvious that

$$q_w + q_o = Q. \tag{13}$$

The fractional flow of water, defined by

$$f_w = \frac{q_w}{Q}, \tag{14}$$

can be expressed as a composite function of S_w through the parameterizations (8)-(10) in the following way. When (3) is added to (4), and simplified, we obtain

$$\frac{\partial P_o}{\partial x}=\frac{-1}{k\left(\dfrac{k_{rw}}{\mu_w}+\dfrac{k_{ro}}{\mu_o}\right)}\left(Q-\frac{kk_{rw}}{\mu_w}\frac{dP_c}{dS_w}\frac{\partial S_w}{\partial x}\right). \tag{15}$$

Further manipulations on (3) yield

$$f_w=\frac{\dfrac{k_{rw}}{\mu_w}}{\dfrac{k_{rw}}{\mu_w}+\dfrac{k_{ro}}{\mu_o}}\left(1+\frac{kk_{ro}}{Q\mu_o}\frac{dP_c}{dS_w}\frac{\partial S_w}{\partial x}\right), \tag{16}$$

in which f_w is expressed as a composite function of $S_w(x, t)$. To obtain a differential equation for the fractional flow f_w, (1) is divided throughout by Q to give

$$\phi\frac{\partial S_w}{\partial t}=-Q\frac{\partial f_w}{\partial x}, \tag{17}$$

which can be regarded as a nonlinear partial differential equation in $S_w(x, t)$. By now, what has taken place is clear: the system of partial differential equations (1)-(4) has been recast into the system (16)-(17) for the purpose of simulation. Henceforth, we shall concentrate on the numerical solution of system (16)-(17).

To complete the specification of the problem, initial and boundary conditions must be given. Without loss of generality, it is assumed that the space coordinates have been rescaled so that the flow domain is always of unit length. Furthermore, water is injected into the finite flow field at the end $x=0$, and ejection of fluids takes place at the end $x=1$. Initially, the flow medium is assumed to be oil-filled, so that the water content is at its minimum allowable value

$$S_w(x,0)=S_{wi}, \qquad 0\leqq x\leqq 1, \tag{18}$$

where S_{wi} is the irreducible water saturation. Since water is injected at $x=0$, the fractional flow of water there must be one. Therefore, the inlet boundary condition is

$$f_w(0,t)=1, \qquad t\geqq 0. \tag{19}$$

As water is uniformly injected into the flow field at $x=0$, only oil is initially produced at the boundary $x=1$. After a while, water begins to accumulate at the out-flow boundary, and it is being held back by the capillary pressure (Leverett (1941)). As time progresses, the water content at $x=1$ increases until the capillary pressure there vanishes, and from that point onwards both water and oil can be ejected simultaneously at $x=1$. Therefore, the outlet boundary conditions are

$$f_w(1,t)=0, \qquad 0\leqq t<t^*, \tag{20}$$

$$S_w(1,t)=S_w^*, \qquad t\geqq t^*, \tag{21}$$

where t^* is the time at which water breakthrough takes place, and S_w^* is the value of water saturation at which the capillary pressure vanishes; that is,

$$P_c(S_w^*)=0. \tag{22}$$

The description of the initial and boundary conditions is now complete.

An implicit block-centered finite-difference scheme with a uniform spatial grid is constructed to solve system (16), (17) numerically. If the number of spatial blocks in the computational algorithm is n, then the length of each block is

$$\Delta x = \frac{1}{n}.$$

Let $S_w(i, j)$ be the water saturation at the center of the ith block at jth time step, and let $f_w(i-\frac{1}{2}, j)$ be the fractional water flow at the interface from the $(i-1)$th block to the ith block, and at the jth time step. Then the discretized equation of flow at the ith interior block is given by

$$S_w(i, j+1) - S_w(i, j) = \frac{Q}{\phi} \frac{\Delta t}{\Delta x}\left(f_w\left(i-\frac{1}{2}, j+1\right) - f_w\left(i+\frac{1}{2}, j+1\right)\right), \tag{23}$$

where $1 \leqq i < n$ and Δt is the time separation between the jth and $(j+1)$th steps. Note that $f_w(i+\frac{1}{2}, j+1) = f_w([(i+1)-\frac{1}{2}], j+1)$, and thus it denotes the water flow at the interface from the ith block to the $(i+1)$th block. The derivation of the finite-difference equations for the boundary blocks are straightforward and will not be dwelt upon. In essence, $S_w(i, j)$ can be obtained for all i and j, and from which other quantities of interest, such as the oil pressure head $P_o(x, t)$ and oil recovery, may be evaluated. Thus, we have now devised a numerical procedure for the simulation of one-dimensional two-phase flow for any given porosity ϕ and absolute permeability k. In the next section, we shall address the issue of the preparation of the porosity and permeability inputs.

3. Heterogeneity and stochasticity. In the traditional, or deterministic, approach to simulation, the porosity ϕ and absolute permeability k are regarded as constants, the values of which are measured by experimentation. For instance, suppose the flow field is a horizontal core. Then the absolute permeability across the core sample, measured in the laboratory, is employed to represent the permeability at each point of the flow domain. Due to heterogeneities or the innate variability of geologic formations, the porosity ϕ and absolute permeability k are not, in general, constant over a given flow field. They should realistically be regarded as spatial stochastic processes $\phi(x)$ and $k(x)$ over the flow domain (Freeze (1975), Smith and Brown (1982), and Ma and Wei (1985)).

There is now a large body of data indicating that at each point of a flow field, permeability has a log-normal distribution and that porosity has a normal distribution (Law (1944) and Freeze (1975)). If permeability is reported in centimeter-squared, it generally lies in the range from 10^{-16} to 10^{-3}, a range covering 13 orders of magnitude (Freeze and Cherry (1979)). In contrast, values of porosity of a specified medium usually varies over a range that is not more than 0.2. Over a short distance, permeability can differ by more than an order of magnitude, whereas for all practical purposes the relative variability of porosity is insignificantly small. Therefore, on a scale in which the variations in permeability are important, perturbations in porosity may be neglected. Henceforth, absolute permeability is taken as a spatial stochastic process with log-normal distribution at each point of the flow domain, and porosity is considered constant. An excellent discussion of the issue of scale in stochastic modeling is given by Vanmarcke (1983), Haldorsen and Lake (1984), and Haldorsen (1985).

A discrete representation of the absolute permeability $k(x)$ as a spatial stochastic process may be obtained in an intuitive manner. Let k_e be the absolute permeability across the flow field in which the displacement process takes place, and let r be the estimated standard deviation in permeability. Both k_e and r can be measured in the laboratory if the flow fields are core samples. Assume that there are n blocks in the uniform spatial partition of the numerical scheme, and denote the absolute permeability of the entire ith block by k_i. The values of the random variable k_i can be generated by the log-normal generator LN (k_e, r), with mean k_e and standard deviation r, for $1 \leqq i \leqq n$. In symbols,

$$k_i \leftarrow \mathrm{LN}\,(k_e, r), \qquad 1 \leqq i \leqq n. \tag{24}$$

Each n-vector $(k_1, k_2, \cdots, k_n)$ thus obtained would adequately serve as a discrete realization of $k(x)$ over the flow domain, provided that the following two constraints are satisfied.

First, there should be internal correlation between the block permeabilities k_i. That means there should be a tendency for a high permeability value to be neighbored by relatively high values, and a low permeability value to be surrounded by relatively low values. Since the block permeabilities k_i are generated by the computer, they are for all practical purposes uncorrelated (Knuth (1969) and Rubenstein (1981)), and a special procedure, to be explained in the next section, is needed to induce statistical correlation. If correlation is ignored, the uncertainty in outputs would change as the number of spatial blocks increases. As a concrete example, take oil pressure head as the output. Suppose the standard deviation $s(x)$ of the oil pressure head is computed at each point of the flow domain. It is found that $s(x)$ varies with n, the number of blocks composing the partition. In general, $s(x)$ diminishes as n increases, or as the partition gets finer (Bouton and Ma (1988)). This anomaly has also been observed in the related studies by Warren and Price (1961) and Freeze (1975). Output uncertainty should only depend on the physical problem and the input uncertainty; it should not be a property of numerical discretization. If spatial correlation is introduced into the random flow field generated by $k(x)$, it is observed that $s(x)$ no longer depends on n provided, of course, that the discretization is fine enough to capture all the significant features of the continuous system. A discussion of this topic is given by Ma, Wei, and Mills (1987).

The second constraint concerns fitting the theoretical model to laboratory data. With the block permeabilities k_i generated by expression (24), the flow domain may be considered as a layered medium. The equivalent permeability of a layered medium is equal to the harmonic mean of the block permeabilities (Greenkorn (1983)). Thus the equivalent permeability across the discretized flow field is the harmonic mean of k_i, $1 \leqq i \leqq n$. This harmonic mean must agree with the permeability value k_e measured across the flow field, and therefore

$$\frac{1}{n}\sum_{i=1}^{n}\frac{1}{k_i}=\frac{1}{k_e}. \tag{25}$$

Even though the k_i are sampled by the generator (24) with mean k_e, condition (25) is not automatically satisfied. In § 5, a scaling method is developed to keep the harmonic mean of the random block permeabilities k_i invariant so that requirement (25) is satisfied. This development is indeed important. Given a specified correlation structure on k_i, the variations of the block permeabilities must be such that their harmonic mean is always equal to the overall permeability k_e measured across the field.

4. Correlation structuring. In mathematical terms, the problem of correlation structuring amounts to devising a mapping $f: \mathbb{R}^n \to \mathbb{R}^n$ such that for every discrete realization $(k_1, k_2, \cdots, k_n) \in \mathbb{R}^n$ of the permeability, the image $f(k_1, k_2, \cdots, k_n) \in \mathbb{R}^n$

possesses a prescribed internal correlation among the elements. For many decades, electrical and control engineers have been tackling the problem of prediction and filtering, and they have devised sophisticated methods for correlating events in time. Information on past events is being drawn upon to predict the present. There is a definite sense of direction in those temporal events: uncertainty always propagates forward into the future. In random field problems there is no such day and night distinction on direction (Whittle (1963)), and there are boundary conditions at both ends. For this reason, techniques such as time series would have only limited success in our current problem (Ma and Wei (1985)). A global procedure that does not rely on any directional properties is needed to correlate the random block permeabilities (Ma, Wei, and Mills (1987)).

Since the permeabilities k_i are log-normally distributed, the variates $\ln k_i$ follow a normal distribution. A complete set of exact relationships between the moments of k_i and those of $\ln k_i$ is available (Vanmarcke (1983) and Ma, Wei, and Mills (1987)), and, as an example,

$$E(k_i^m) = \exp\left(mE(\ln k_i) + \tfrac{1}{2}m^2 \operatorname{Var}(\ln k_i)\right), \tag{26}$$

where $E(\bullet)$ is the expectation operator and $\operatorname{Var}(\bullet)$ the variance operator. For this reason, it is permissible to use $\{k_i\}$ or $\{\ln k_i\}$ interchangeably. Hence, we shall construct a specific correlation structure on $\{\ln k_i\}$ instead. This amounts to devising a mapping $g:\mathbb{R}^n \to \mathbb{R}^n$ such that for every $(\ln k_1, \ln k_2, \cdots, \ln k_n) \in \mathbb{R}^n$, the image $g(\ln k_1, \ln k_2, \cdots, \ln k_n) \in \mathbb{R}^n$ possesses a specified correlation structure. The relationship between the mappings f and g, as illustrated in Fig. 1, is

$$f = \exp \circ g \circ \ln, \tag{27}$$

where $\circ$ is the composition of mappings. In other words, $f(\cdot) = \exp(g(\ln(\cdot)))$. The above expression demonstrates, for one more time, the duality of correlation structuring on $\{k_i\}$ and $\{\ln k_i\}$. Let us proceed to discuss how values of the random vector $g(\ln k_1, \ln k_2, \cdots, \ln k_n)$ can be generated.

As explained in (24), the variates k_i are log-normally distributed with mean k_e and standard deviation r. Therefore, the variates $\ln k_i$ have a normal distribution with mean μ for which

$$\mu = \ln\left\{\frac{k_e^2}{(r^2 + k_e^2)^{1/2}}\right\}. \tag{28}$$

$$\begin{tikzcd}
(\ln k_1, \ldots, \ln k_n) \arrow[r, "g"] & g(\ln k_1, \ldots, \ln k_n) \arrow[d, "\exp(\bullet)"] \\
(k_1, \ldots, k_n) \arrow[u, "\ln(\bullet)"] \arrow[r, "f"] & f(k_1, \ldots, k_n)
\end{tikzcd}$$

FIG. 1. *Commutative diagram for mappings f and g.*

Suppose G is a given positive semidefinite matrix and it is specified that $\{\ln k_i\}$ should have G as the covariance matrix. Then samples of the random vector $g(\ln k_1, \ln k_2, \cdots, \ln k_n)$, denoted by $(x_1, x_2, \cdots, x_n)$, can be generated directly by multivariate statistical techniques. First choose normal random values z_i with zero mean and unit variance from the standard normal generator $N(0, 1)$. In symbols,

$$z_i \leftarrow N(0,1), \qquad 1 \leqq i \leqq n. \tag{29}$$

Then the values of x_i are given by

$$x_i = \mu + \sum_{j=1}^{n} l_{ij} z_j, \qquad 1 \leqq i \leqq n, \tag{30}$$

where the l_{ij} are the components of the lower triangular matrix L defined by the equation

$$G = LL^T. \tag{31}$$

The process of computing the l_{ij} is simply the process of forming the Cholesky factorization of G, which is straightforward if G is positive definite (Martin, Peters, and Wilkinson (1965)). In the unusual case that the semidefinite matrix G is not positive definite, then some of the eigenvalues of G would vanish, and the problem may be reduced to one of lower dimensions (Ma, Wei, and Mills (1987)). Hence, it is assumed that expression (30) can always be employed to produce components x_i of n-vectors with mean μ and specified covariance matrix G.

To demonstrate the effect of the correlation filter (30), random vectors of dimension 50 are selected by the standard normal generator. Each vector is then laid on the unit interval by representing its elements as vertical coordinates of 50 equidistant points on the interval. Two such random vectors are plotted in Fig. 2, where the lack of

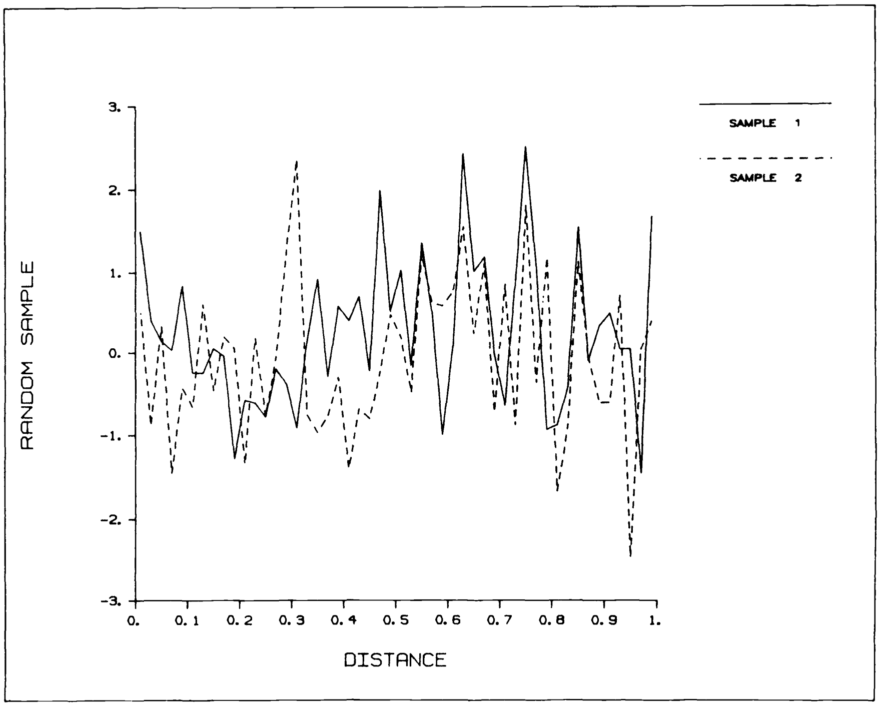


FIG. 2. *The uncorrelated sample elements of random 50-vectors.*

internal correlations of the components manifests itself as erratic fluctuations. Suppose the elements of the 50-vectors are to be homogeneously correlated so that the correlation function $\rho(x)$ is

$$\rho(x) = \exp\left(-\frac{1}{\lambda}|x|\right), \tag{32}$$

where x denotes the separation between the elements and λ is a specified parameter. The corresponding covariance matrix G is of order 50 whose (i, j)th element g_{ij} is given by

$$g_{ij} = \exp\left(-\frac{1}{50\lambda}|i-j|\right), \tag{33}$$

for $1 \leqq i, j \leqq 50$. This positive definite matrix can then be decomposed in the form (31). Since

$$\lambda = \int_0^\infty \rho(x)\, dx, \tag{34}$$

the parameter λ may be thought of as the average correlation length. Set $\lambda = 0.1$. The effect of passing the two sample vectors of Fig. 2 through the filter (30) is shown in Fig. 3. For the two filtered sample vectors of Fig. 3, there is a tendency for large elements to be surrounded by relatively large elements, and abrupt fluctuations have been greatly smoothed down, even though the correlation length is only 0.1. Transforming the two samples of Fig. 2 by a stronger filter, with $\lambda = 0.5$, results in more highly structured output sample vectors, as shown in Fig. 4. Similar experiments with covariance matrices of other forms lead to the identical observation that correlation structuring plays a significant role in stochastic simulation.

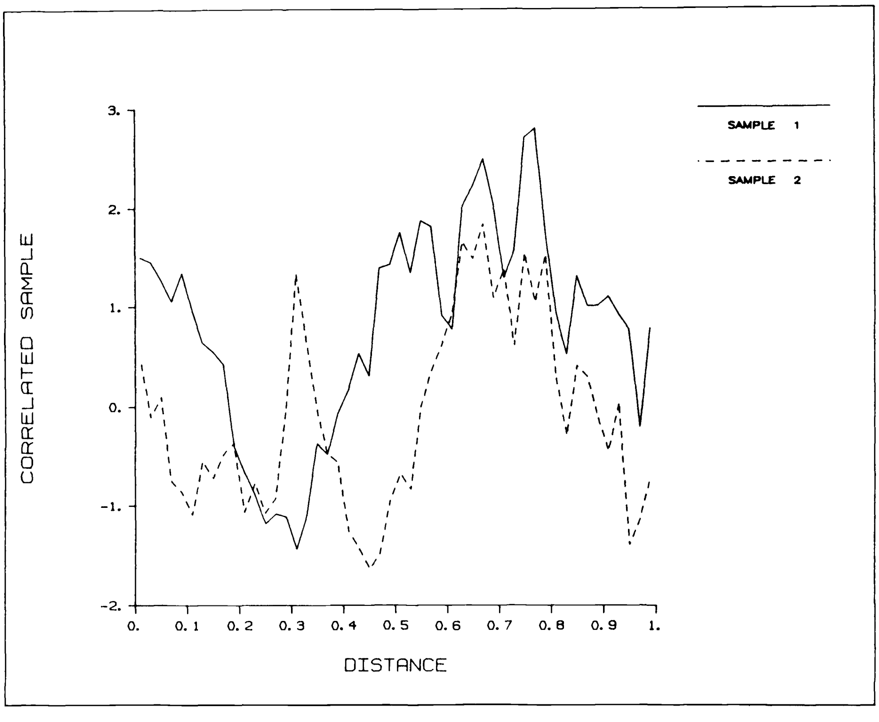


FIG. 3. *The internally correlated sample elements of filtered random vectors, with correlation length of* 0.1.

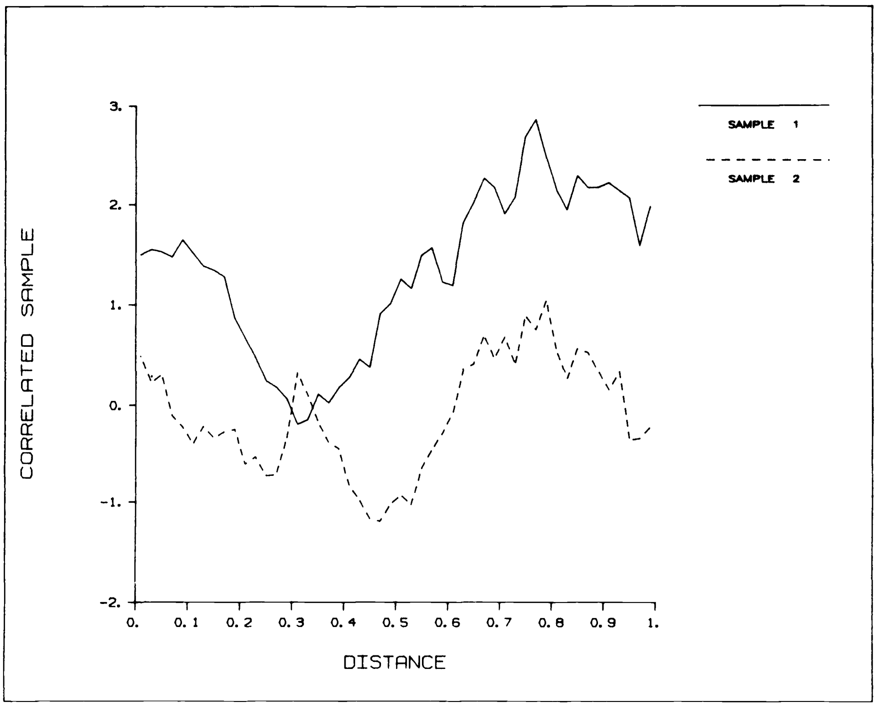


FIG. 4. *The internally correlated sample elements of filtered random vectors, with correlation length of* 0.5.

It has been pointed out in the last section that correlation structuring stabilizes the outputs of stochastic simulation. Suppose the correlation structure to be built into the block permeabilities is given by (32), and suppose the number of blocks n in the uniform partition is already large enough to ensure the usual numerical convergence. To capture the significant features of correlation, it is also required that the block size be smaller than the correlation length λ. Although a theory on this topic does not exist, the numerical experiments conducted by Bouton and Ma (1988) in a different problem indicate that the largest block size should not generally exceed $\lambda/2$, so that standard deviations of outputs would not change as the number of blocks n is increased. Assuming that the flow domain has been rescaled to have unit length, then

$$\frac{1}{n} < \frac{\lambda}{2}$$

for stochastic numerical stability. Of course, n should already be large enough to ensure the deterministic numerical convergence.

5. Invariance of harmonic mean. In earlier sections, a procedure has been given to construct permeability n-vectors such that the elements have a log-normal distribution with mean k_e and standard deviation r and a prescribed internal correlation relationship. As explained in expression (25), a device is needed to ensure that the harmonic mean of the block permeabilities is equal to k_e. When this is accomplished, our stochastic model would agree, in all input data, with those measured by experimentation.

Let $(k_1, k_2, \cdots, k_n)$ be a permeability n-vector such that k_i is LN (k_e, r) for each i, and the permeabilities k_i satisfy a prescribed internal correlation. A topological scaling $h: \mathbb{R}^n \to \mathbb{R}^n$ is a mapping on the permeability n-vector such that the image

$h(k_1, k_2, \cdots, k_n)$ satisfies constraint (25). To find one possible h, let us adopt the straightforward construction

$$h(k_1, k_2, \cdots, k_n) = (sk_1, sk_2, \cdots, sk_n), \tag{35}$$

where s is a random variable for which

$$\frac{1}{n}\sum_{i=1}^{n}\frac{1}{sk_i} = \frac{1}{k_e}. \tag{36}$$

The above transformation will distort the correlation relationship that has been built into the block permeabilities k_i. To investigate the effect of this distortion, we proceed in the following way. Without the scaling factor s, the block permeabilities satisfy

$$\frac{1}{n}\sum_{i=1}^{n}\frac{1}{k_i} = \frac{1}{k_H}, \tag{37}$$

where k_H is a random variable. Define

$$v^2 = \ln\left(1 + \frac{r^2}{k_e^2}\right). \tag{38}$$

It is claimed that s converges to the deterministic constant $\exp(v^2)$, as n increases without bound. To see this, k_i is LN (k_e, r) implies that $\ln k_i$ is $N(\mu, v)$, where μ and v are defined by (28) and (38), respectively. This, in turn, implies that $-(\ln k_i) = \ln(1/k_i)$ is $N(-\mu, v)$, and therefore $1/k_i$ follows a log-normal distribution with mean $\exp(-\mu + \frac{1}{2}v^2)$, in agreement with (26). Now the right-hand side of expression (37) consists of the sum of n weakly correlated random variables, none of which is dominant, and each has the same mean and variance. By the generalized Central Limit Theorem, this sum tends, as n increases, to a normal random variable with mean $\exp(-\mu + \frac{1}{2}v^2)$ and with variance decreasing in the order $1/n$. Therefore, $1/k_H$ tends to the deterministic constant $\exp(-\mu + \frac{1}{2}v^2)$, as $n \to \infty$. From (36), however,

$$\frac{1}{n}\sum_{i=1}^{n}\frac{1}{sk_i} = \frac{1}{sk_H} = \frac{1}{k_e}. \tag{39}$$

In other words,

$$s = \frac{k_e}{k_H}. \tag{40}$$

Thus, as $n \to \infty$, s tends to $k_e \exp(-\mu + \frac{1}{2}v^2)$. But from (28) and (38)

$$\begin{aligned} k_e \exp\left(-\mu + \frac{1}{2}v^2\right) &= k_e\left\{\frac{(r^2 + k_e^2)^{1/2}}{k_e^2}\right\}\exp\left(\frac{1}{2}v^2\right) \\ &= \left(1 + \frac{r^2}{k_e^2}\right)^{1/2}\exp\left(\frac{1}{2}v^2\right) \\ &= \exp(v^2). \end{aligned} \tag{41}$$

Therefore

$$s \to \exp(v^2) \quad \text{as } n \to \infty. \tag{42}$$

For very slight heterogeneities, the standard deviation r of the block permeabilities would be very small, and so would v^2, by virtue of expression (38). In this case, the asymptotic value of the scaling factor s is practically one. Several computer experiments are conducted to study the convergence of expression (42). It has been found, using stratified sampling techniques, that the initial convergence is generally rapid. For weak heterogeneities and for correlation lengths of about 10 percent, s can be very close to its limiting value when $n \geqq 20$.

It is now clear that the mapping (35) scales any covariance structure of the block permeabilities by an asymptotic factor of $\exp(2v^2)$. This distortion can be easily accounted for by specifying an initial covariance relationship which is $\exp(-2v^2)$ times the desired values. Hence, the invariance of the harmonic mean of the block permeabilities can be maintained while preserving a specified correlation structure.

6. Simulations of transient flow. The results of a computer-synthesized experiment on waterflooding in a horizontal one-dimensional medium will be reported in this section, principally to study the effect of stochastic perturbations of permeability on oil pressure and recovery. The construction of the Monte Carlo algorithm has been based on the theory expounded in previous sections. Two minor additions not explained earlier are now brought up. First, to fully accommodate the heterogeneous variations of the absolute permeability, the J-function (Leverett (1941) and Amyx and Bass, Jr. (1962)) is employed to impart to the capillary pressure spatial variability. The capillary pressure P_{ci} at the ith block is computed by

$$P_{ci} = \left(\frac{k_e}{k_i}\right)^{1/2} P_c, \tag{43}$$

where k_i is the absolute permeability of the ith block, and P_c is given by expression (10). The second addition is the incorporation of variance reduction methods. Since the Monte Carlo method involves the repetitive evaluations of system parameters, special techniques may be applied to streamline the computations. To economize on both core memory and computing time, the methods of antithetic variates and intermediate control variates (Hammersley and Morton (1956), Ang and Tang (1984), and Bratley, Fox, and Schrage (1983)) are employed. However, the application of these variance reduction techniques merely improves upon the efficiency of the numerical algorithm, and they do not constitute an essential component to the algorithm.

The input data for the synthetic experiment are as follows. The flow domain consists of a cylindrical core of length $L = 30.16$ cm, cross-sectional area $A = 11.40$ cm^2, and porosity $\phi = 26.4$ percent. Since one-dimensional flow is studied, the geometry of the cross section of the core is not needed for calculation. The injection rate $Q =$ 4.08 ft/day, the viscosity of oil $\mu_o = 0.975$ cp, and viscosity of water $\mu_w = 0.33$ cp. The absolute permeability measured across the core $k_e = 82$ mD, with estimated standard deviation $r = 26.6$ mD. From our earlier discussion of the range of variability of absolute permeability, this value of r represents truly moderate input uncertainty. For the given core material, the irreducible water saturation $S_{wi} = 0.488$ and the residual oil saturation $S_{or} = 0.281$. The constants p_i involved in the parameterizations (8)-(10) are $p_1 = p_3 = p_5 = p_7 = 0$; $p_2 = 1.58$; $p_4 = 0.774$; $p_6 = 0.793$; $p_8 = 0.0817$; and $p_9 = 0.662$. These values of p_i are expert guesses supplied by engineers who have worked fairly extensively with cores extracted from the same formation.

As mentioned earlier, the flow domain is always rescaled to have unit length, and injection of water always takes place at $x = 0$. A uniform partition of $n = 21$ blocks is used, and an exponential correlation (32) with length $\lambda = 0.1$ is induced upon the block permeabilities. A total of $M = 300$ Monte Carlo passes is employed for this study. Both the values of M and n have been varied slightly, and it is found that the output data do not change appreciably. That means the choices of $M = 300$ and $n = 21$ would probably be sufficient to capture all the essential characteristics of the heterogeneities. It is not easy, however, to analytically investigate the stochastic stability and convergence of the outputs (Bendat and Piersol (1971) and Ma, Wong, and Caughey (1983)). In interpreting the output data, only the means and standard deviations are highlighted. And, when temporal variations of the results are presented, the time variable is always expressed in terms of pore volume injected. The complete numerical experiment consumes about 70 min in CPU time on a CYBER 205 computer. We have been informed that if we were to vectorize our codes, the CPU time could be reduced by up to an order of magnitude.

At the beginning of the synthetic experiment the core is assumed to be oil-filled. As water is progressively injected into the core at $x = 0$, oil is ejected at $x = 1$. The variation of the mean oil pressure head is shown in Fig. 5, where the time is expressed in terms of pore volume injected. At any fixed point along the core, the peak value of the average $P_o(x, t)$ occurs at the time of water breakthrough. On the other hand, the average $P_o(x, t)$ decreases almost linearly over the flow domain at any time after breakthrough, as is expected. The space-time distribution of the 125th realization of the pressure head $P_o(x, t)$ is shown in Fig. 6, which gives a snapshot of the effect of heterogeneity on pressure. In Fig. 7, the variation of the standard deviation of $P_o(x, t)$ is exhibited, from the side near $x = 1$, while Fig. 8 displays the same distribution from the side near $x = 0$. At early times the maximum uncertainties in $P_o(x, t)$ occur near the boundaries where large instantaneous drops in pressure head have taken place, while at later times they shift to the central portions of the flow field, consistent with steady-state results (Ma, Wei, and Mills (1987)). The uncertainties in $P_o(x, t)$ vanish along the boundary $x = 1$, because the pressure is held constant there. In general, it is not uncommon for the standard deviation of oil pressure head to exceed 50 percent of the corresponding mean values, a degree of variability substantially larger than what engineers used to assume on oil pressure.

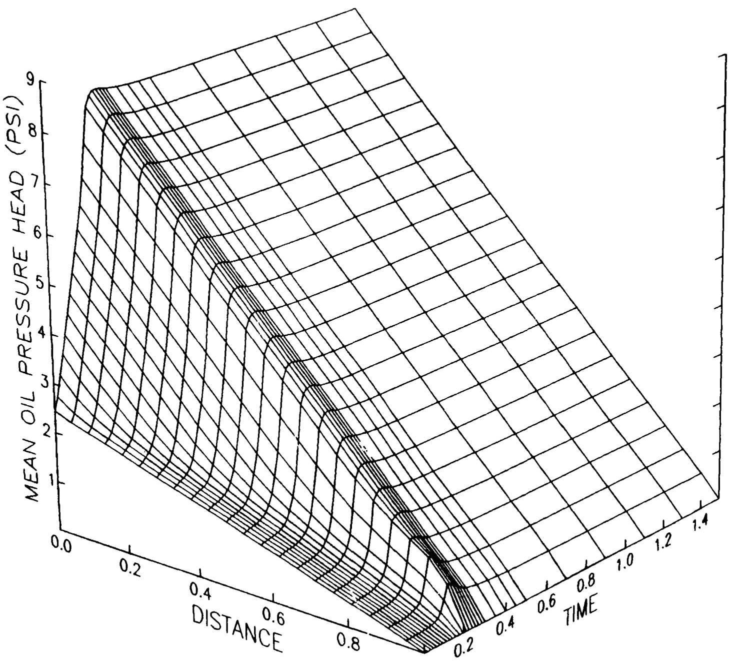


FIG. 5. *Variation of the mean oil pressure head in space and time.*

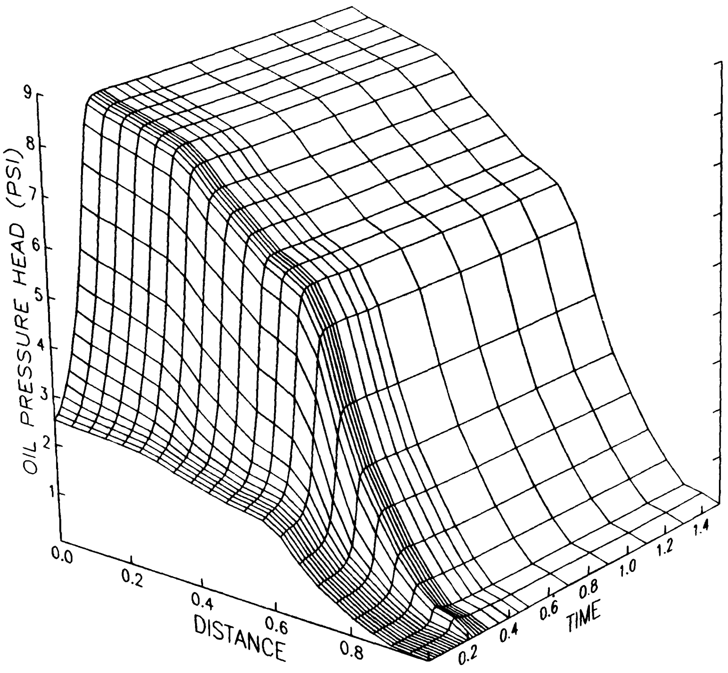


FIG. 6. *Variation of the 125th realization of oil pressure head.*

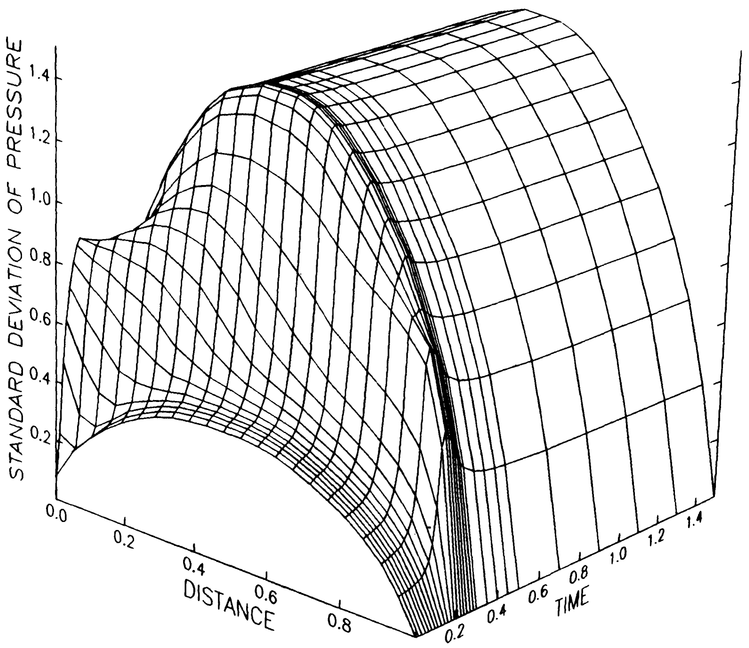


FIG. 7. *Variation of the standard deviation of oil pressure, from the side of ejection.*

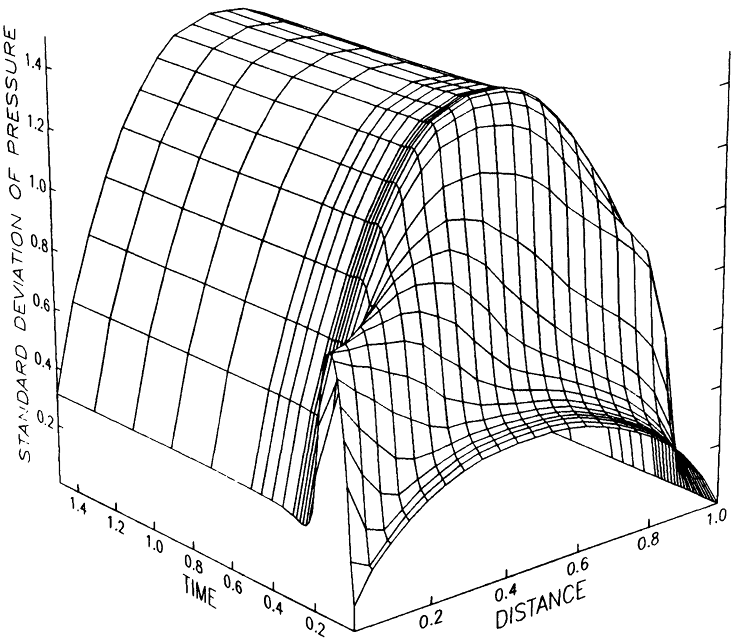


FIG. 8. *Variation of the standard deviation of oil pressure, from the side of injection.*

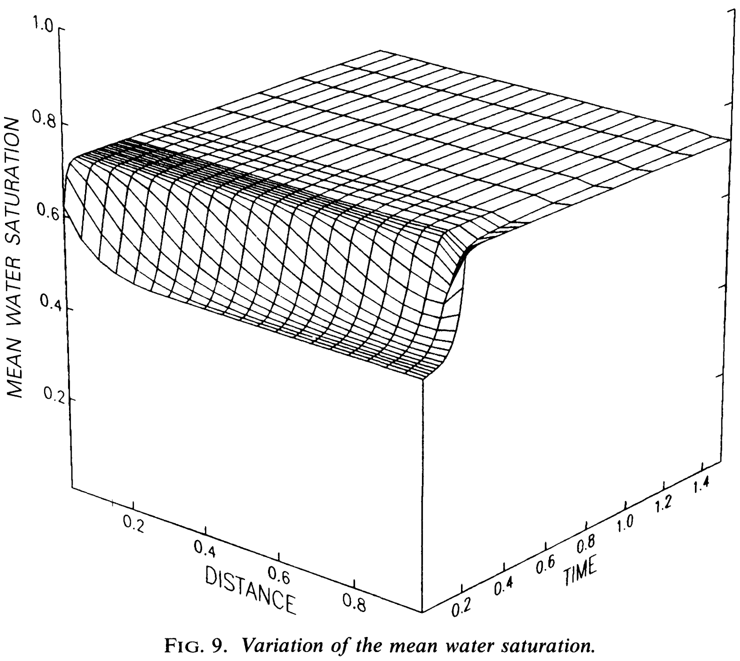


FIG. 9. *Variation of the mean water saturation.*

To further visualize the evolution of the waterflooding experiment, the variation of the average water saturation is shown in Fig. 9, where the drop in saturation near $x=1$ is due to capillary end effects (Leverett (1941)). The pressure drops across the core have been measured in the laboratory and are shown in Fig. 10. All data points recorded in the laboratory are indicated by star markers. For the purpose of comparison, the computed mean pressure drops and the corresponding standard deviation is also displayed in Fig. 10. In Fig. 11, the computed mean oil recovery is compared to laboratory data. Also shown is the standard deviation of the oil recovery, which is rather small. In general, the agreement between simulation outputs and experimental data is fairly close.

Finally, a deterministic run has been made with k_e as the constant permeability input. The oil pressure head obtained from this deterministic run is compared with the mean oil pressure head of Fig. 5, and the differences are less than 6 percent everywhere. This shows that we have not gone too far afield in our theoretical development. Several other computer-synthesized waterflooding experiments have been conducted (Ma and Wei (1985)), and it is found that the results presented in this section are indeed representative.

7. Sensitivity of relative permeabilities. In the last section, it has been observed that oil pressure head $P_o(x, t)$ can exhibit variability substantially larger than what engineers used to assume. Naturally, we would like to know the cause. Since the parameterizations (8) and (9) for k_{ro} and k_{rw} are empirical in nature, it is suspected that these parameterizations might be overly sensitive to heterogeneous variations. To

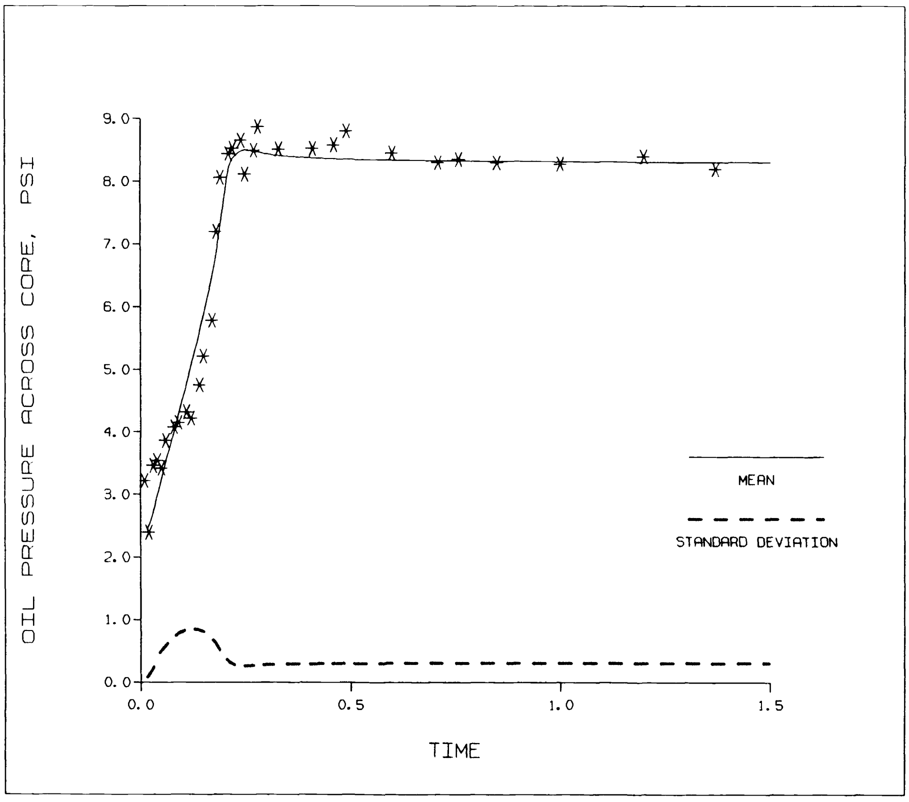


FIG. 10. *Theoretical pressure head as compared to experimental data points.*

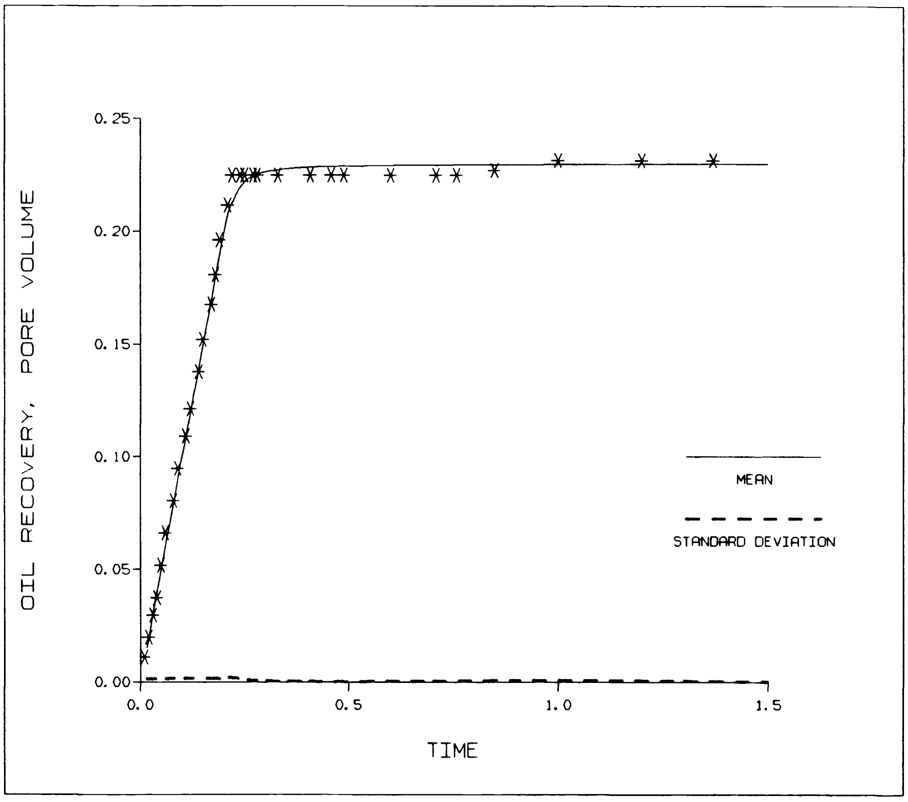


FIG. 11. *Theoretical recovery as compared to experimental data points.*

look into this question, we have adopted least-squares regression analysis to solve the inverse problem of how k_{ro} and k_{rw} are perturbed by the heterogeneities k_i. Given the inputs k_{ro} and k_{rw}, a total of $M = 300$ output realizations has been generated in the previous section. Among these outputs are the oil pressure drops across the flow field and the oil recovery. If heterogeneous variations of the absolute permeability were absent, so that $k_i = k_e$ for all i, then it would require 300 sets of possibly different inputs of k_{ro} and k_{rw} to deterministically generate those 300 sets of pressure drops and oil recovery data. The least-squares estimator is the device that we can use to compute the 300 sets of fictitious k_{ro} and k_{rw} inputs (Lasdon et al. (1978)). However, an enormous amount of CPU time is needed for the least-squares calculation, and it becomes impermissible to pass all 300 stochastic outputs through the least-squares estimator. To compromise, 50 samples are randomly selected from the 300 sets of oil pressure and recovery data, and passed into the least-squares estimator. The largest and the smallest k_{ro} and k_{rw} from the least-squares estimator, as well as the original k_{ro} and k_{rw} that have been used to generate the 300 sets of oil pressure and recovery, are all shown in Fig. 12. With 50 samples, the differences between the largest and smallest k_{ro} and k_{rw} are fairly small, indicating that the parameterizations (8) and (9) for relative permeabilities are rather insensitive to heterogeneities.

8. Conclusions. In this paper an algorithmic formulation has been given for the modeling of heterogeneities in linear two-phase flow. A stochastic approach has been adopted, whereby the absolute permeability is taken as a spatial stochastic process with a log-normal distribution at each point of the flow domain. A correlation structure

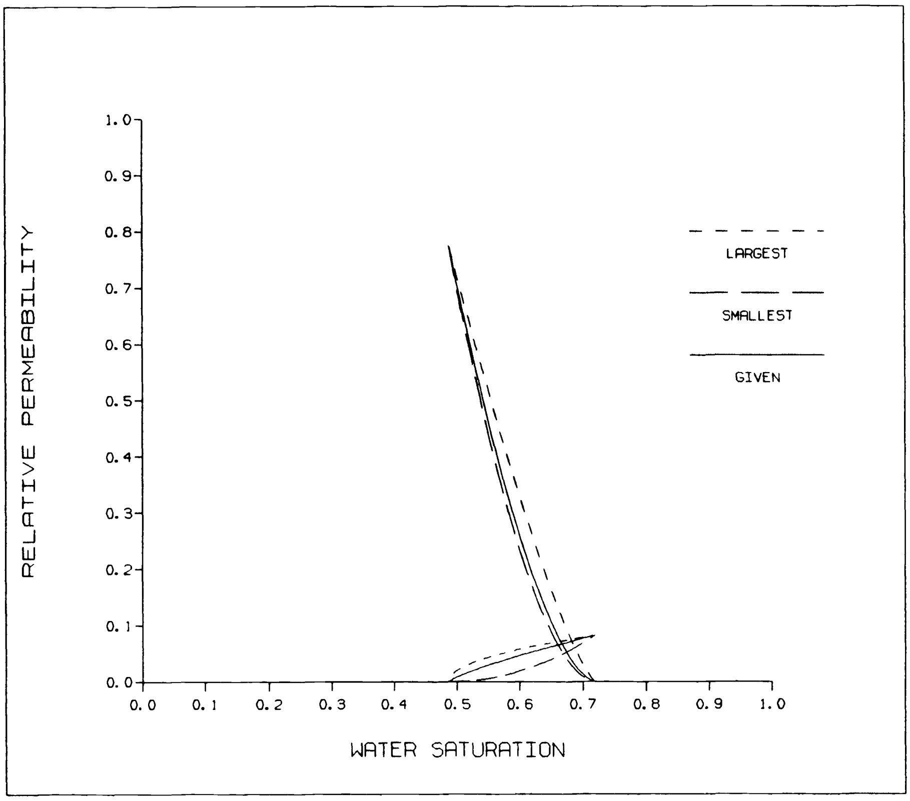


FIG. 12. *Sensitivity of relative permeabilities.*

has been built into the set of stochastic variables so as to achieve a realistic representation. In addition, a constraint on the invariance of harmonic mean has been attained by way of topological scaling. A computer-synthesized waterflooding experiment has been conducted to explore the effect of heterogeneity on oil pressure head and recovery. Limited comparison with laboratory data has also been made. By employing a least-squares estimator, the sensitivity of the empirical parameterizations of relative permeabilities has also been qualitatively investigated.

Three statements are in line. First, it has been observed that output uncertainties can become significant, even when the heterogeneities are moderate. It appears reasonable, therefore, to suggest that deterministic or averaged models for linear two-phase flow would not be adequate for the purpose of reliability-based designs. Second, it has been seen, by working with only a small number of samples, that parameterizations of relative permeabilities are not highly sensitive to heterogeneities. If the large output uncertainties are caused by oversensitivity of some system parameters, the relative permeabilities are probably not among them. Third, the Monte Carlo scheme devised in this paper may be used in higher dimensions.

The above three statements have been based upon observations made with one-dimensional simulations. These observations are only as credible as the assumptions underlying the model. In higher dimensions, two-phase flow is much more complex. There is the possibility of channeling past low-permeability elements in two-dimensional flow. This localization of the effect of low-permeability elements would moderate the influence of heterogeneity on the flow. Fingering instability can arise in two-dimensional flow, and this would cause large decrease in production. Also, as explained by Whittle (1963), it is not clear how elementary correlation functions can be specified in two dimensions. While we believe that observations made in this investigation are both important and inspiring, it must be emphasized that these observations are only as reliable as the assumptions underlying the one-dimensional model. Finally, algorithmic techniques expounded herein may be readily modified and applied to other areas of science and engineering.

Appendix.

NOMENCLATURE

S_w = water saturation
S_o = oil saturation
A = cross-sectional area of core
L = length of core
Q = total flow rate per unit area
ϕ = porosity
μ_w = water viscosity
μ_o = oil viscosity
P_w = water pressure
P_o = oil pressure
S_{wi} = irreducible water saturation
S_{or} = residual oil saturation
S = normalized water saturation
S_w^* = value of S_w for which $P_c = 0$
k_{rw} = relative permeability of water
k_{ro} = relative permeability of oil
λ = fractional correlation length
k_e = absolute permeability measured across flow field
k_i = absolute permeability of ith block
M = number of Monte Carlo passes
n = number of space blocks in numerical algorithm
P_c = capillary pressure
P_{ci} = capillary pressure at ith block
r = standard deviation of absolute permeability
s = standard deviation of oil pressure head

REFERENCES

J. W. AMYX AND D. W. BASS, JR. (1962), *Properties of Reservoir Rocks, Petroleum Production Handbook*, Vol. II, Reservoir Engineering, T. C. Frick, ed., SPE, Dallas, TX.

M. L. ANDERSON (1979), *Application of risk analysis to enhanced recovery pilot testing decisions*, J. Pet. Tech., 31, pp. 1525–1530.

A. H.-S. ANG AND W. H. TANG (1984), *Probability Concepts in Engineering Planning and Design*, Vol. II, Decision, Risk, and Reliability, John Wiley, New York.

J. BEAR (1972), *Dynamics of Fluids in Porous Media*, Elsevier, New York.

J. S. BENDAT AND A. G. PIERSOL (1971), *Random Data: Analysis and Measurement Procedures*, Interscience, New York.

P. M. BOUTON AND F. MA (1988), *On spatial dependence in Monte Carlo simulations of random fields*, Internat. J. Modeling Simulation, 8, pp. 94–97.

P. Bratley, B. L. Fox, and L. E. Schrage (1983), *A Guide to Simulation*, Springer-Verlag, Berlin, New York.

T. K. Caughey (1971), *Nonlinear theory of random vibrations*, in Advances in Applied Mechanics, Vol. 11, C. S. Yih, ed., pp. 209-253, Academic Press, New York.

J. D. Collins and J. M. Hudson (1981), *Applications of risk analysis in nuclear structures*, Proc. Symposium Problem Methods in Structural Engineering, M. Shinozuka and J. T. P. Yao, eds., American Society of Civil Engineers, pp. 153-178.

H. Contreras (1980), *The stochastic finite-element method*, Comput. & Structures, Vol. 12, pp. 341-348.

J. Douglas, P. M. Blair, and R. J. Wagner (1958), *Calculation of linear waterflood behavior including the effects of capillary pressure*, Pet. Trans. AIME, 213, pp. 96-102.

A. S. Fokas and Y. C. Yortsos (1982), *On the exactly solvable equation occurring in two-phase flow in porous media*, SIAM J. Appl. Math., 42, pp. 318-332.

R. A. Freeze (1975), *A stochastic-conceptual analysis of one-dimensional groundwater flow in nonuniform homogeneous media*, Water Resour. Res., 11, pp. 725-741.

R. A. Freeze and J. A. Cherry (1979), *Groundwater*, Prentice-Hall, Englewood Cliffs, NJ.

R. A. Greenkorn (1983), *Flow Phenomena in Porous Media*, Marcel Dekker, New York.

H. H. Haldorsen (1985), *Simulator parameter assignment and the problem of scale in reservoir engineering*, SPE Conference on Reservoir Characterization, Dallas, TX.

H. H. Haldorsen and L. W. Lake (1984), *A new approach to shale management in field-scale models*, Soc. Pet. Eng. J., 24, pp. 447-457.

J. M. Hammersley and K. W. Morton (1956), *A new Monte Carlo technique: Antithetic variables*, Proc. Cambridge Phil. Soc., 52, pp. 449-475.

D. E. Knuth (1969), *The Art of Computer Programming*, Vol. 2, Seminumerical Algorithms, Addison-Wesley, Reading, MA.

L. S. Lasdon, A. D. Waren, A. Jain, and M. Ratner (1978), *Design and testing of a generalized reduced gradient code for nonlinear programming*, ACM Trans. Math. Software, 4, pp. 34-50.

J. Law (1944), *A statistical approach to the interstitial heterogeneity of sand reservoirs*, Pet. Trans. AIME, 155, pp. 202-222.

M. C. Leverett (1941), *Capillary behavior in porous solids*, Pet. Trans. AIME, 142, pp. 152-169.

D. Ludwig (1975), *Persistence of dynamical systems under random perturbations*, SIAM Rev., 17, pp. 605-640.

F. Ma, F. S. Wong, and T. K. Caughey (1983), *On the Monte Carlo methodology for cumulative damage*, Computers and Structures, 17, pp. 177-181.

F. Ma and M. S. Wei (1985), *Stochastic Interpretation of Linear Corefloods*, Report No. RE18FM1, The Standard Oil Company, Cleveland, OH.

F. Ma, M. S. Wei, and W. H. Mills (1987), *Correlation structuring and the statistical analysis of steady-state groundwater flow*, SIAM J. Sci. Statist. Comput., 8, pp. 848-867.

R. S. Martin, G. Peters, and J. H. Wilkinson (1965), *Symmetric decomposition of a positive definite matrix*, Numer. Math., 7, pp. 362-383.

M. Muskat (1949), *Physical Principles of Oil Production*, McGraw-Hill, New York.

W. H. Rowan, N. Lipner, J. J. Farrell, J. D. Oliver, and A. S. Nii (1974), *Failure Analysis by Statistical Techniques*, Vol. I, DNA 3336F-1, TRW Systems Group, Redondo Beach, CA.

R. Y. Rubenstein (1981), *Simulation and the Monte Carlo method*, John Wiley, New York.

L. Smith and R. A. Freeze (1979), *Stochastic analysis of steady state groundwater in a bounded domain. 1. One-dimensional simulations*, Water Resour. Res., 15, pp. 521-538.

P. J. Smith and C. E. Brown (1982), *Stochastic modeling of two-phase flow in porous media*, 57th SPE Annual Technical Conference, New Orleans, LA.

E. H. Vanmarcke (1983), *Random Fields: Analysis and Synthesis*, MIT Press, Cambridge, MA.

J. E. Warren and H. S. Price (1961), *Flow in heterogeneous porous media*, Soc. Pet. Eng. J., 1, pp. 153-169.

P. Whittle (1963), *Stochastic Processes in Several Dimensions*, Bull. Int. Stat. Inst., 40, pp. 979-994.

J. L. Willems (1975), *Stability Criteria for Stochastic Systems with Colored Multiplicative Noise*, Acta Mech., 23, pp. 171-178.

M. R. J. Wyllie (1962), *Relative Permeability, Petroleum Production Handbook*, Vol. II, Reservoir Engineering, T. C. Frick, ed., SPE, Dallas, TX.

Y. C. Yortsos and A. S. Fokas (1983), *An analytical solution for linear waterflood including the effects of capillary pressure*, Soc. Pet. Eng. J., 23, pp. 115-124.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 6, pp. 1073–1086, November 1990

© 1990 Society for Industrial and Applied Mathematics
003

FAST HYBRID SOLUTION OF ALGEBRAIC SYSTEMS*

CRAIG C. DOUGLAS†, JAN MANDEL‡, AND WILLARD L. MIRANKER†

Abstract. The error and timing of solvers consisting of both analog and digital circuitry for sparse linear systems of equations are proposed and analyzed. High speed but low precision is obtained from the analog circuits. This is combined with low speed but high precision from the digital circuits. The hybrid circuit should be faster than digital circuits alone. As a preconditioner to standard iterative solution methods, the hybrid circuit makes the cost of the preconditioning step negligible. The hybrid circuit is also applied to a standard multilevel algorithm.

Key words. digital-analog computing, linear systems of equations, multigrid methods

AMS(MOS) subject classifications. 65F05, 65F10

1. Introduction.

We study a fast equation solver consisting of both analog and digital circuitry. We expect this hybrid combination to give better results than digital techniques alone. The basic idea is to use an analog solver as a preconditioner for a digital iterative process. For a related study, see [6]. Thus, we can obtain both high speed from a fast exchange of information in analog circuitry and high precision from digital circuitry. Eventually, both types of circuits should be integrated onto a single chip.

In § 2, we define an analog defect correction algorithm and discuss the sources of error. We also provide an error analysis. In § 3, we define a basic model for a simple analog solver. We analyze its response speed and its precision. A general example is examined in detail. In § 4, we define and motivate a two-stage analog solver. As in § 3, we analyze it and examine an example. Finally, in § 5, we define and analyze a multilevel solver. We use the term multilevel in the abstract multigrid solution of partial differential equation sense.

The applicability of the method is essentially limited by the condition $\varepsilon\kappa \ll 1$, where ε is the relative precision of the analog circuitry and κ is the condition number of the linear system. For the multilevel solver, the condition number involved is the one for the linear system on the coarsest level. The time required for the analog part of the method also depends on the condition number. We conclude that this time is negligible in comparison to that for the digital part of the method when $\varepsilon\kappa \ll 1$.

Due to ε being technology dependent, the limit of this theory is currently $\kappa \leqq 1{,}024$. A sequel to this paper [10] considers preconditionings and modifications to the analog-digital algorithm in § 2 to apply this theory to problems where $\kappa > 1{,}024$.

We expect the principal benefits of the proposed method to manifest themselves with advances in technology: analog circuitry has the potential to avoid the information exchange bottleneck of massively parallel digital computation. Essentially, we are trading (recoverable) precision for fast dissemination of information between the processors.

We expect that these techniques will be advantageous for large but moderately conditioned positive definite problems with well-defined sparsity structures. Systems arising by either finite-element or finite-difference discretizations of partial differential

* Received by the editors June 13, 1988; accepted for publication (in revised form) September 1, 1989.

† Mathematical Sciences Department, IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.

‡ Computational Math Group, Department of Mathematics, University of Colorado, Denver, Colorado 80202.

equation problems are one possible application. For a general, nonsparse system, the number of connections required is prohibitive.

The technique used in the analysis is classical and can be found in any electrical engineering textbook. We believe the defect correction approach to the hybrid method is new and offers as yet unexplored possibilities in massive parallel computation. For related studies, see [1], [3], and [8].

The precision of current analog circuity is up to 10 bits [12] using capacitors as basic circuit components. Optical processors have the same or lower precision [11]. A purely optical analog method for solving linear systems is presented in [5]. See [7] for a survey of basic concepts of optical computing.

2. Analog defect correction: Error analysis.

Consider a system of linear equations in matrix notation:

$$Ax^* = f,$$

where A is an $n \times n$ nonsingular matrix and x^* is the exact solution. We will sometimes require the hypothesis that A is symmetric, positive definite. In the following algorithm, we use a standard residual correction technique to solve the system, except that part of the computation is performed digitally and part is performed with an analog solver.

THE HYBRID ALGORITHM (Iterative).

```
Step 1. For given $x_1$, computer $r = f - Ax_1$, digitally.
Step 2. Convert $r$ to $n$ parallel analog signals, and using an analog solver, solve
        the equation
                $Ay = r.$
        Convert $y$ (in parallel) to digital output.
Step 3. Compute $x_2 = x_1 + y$, digitally.
Step 4. Set $x_1 \leftarrow x_2$ and go to Step 1.
```

We can analyze the error reduction per step using techniques found in [13]. Assume that the precision of digital computation is very high relative to the analog computation. The quality of the analog computation is described by the following backward characterization:

$$y = (A - E)^{-1} r + e, \tag{1}$$

where E is a perturbation matrix and e a perturbation vector such that

$$\|e\| \leqq \varepsilon \|(A - E)^{-1} r\| \tag{2}$$

and

$$\|E\| \leqq \varepsilon \|A\|. \tag{3}$$

Here, $\|\cdot\|$ is a suitable norm (the l_∞ norm will be appropriate here) and ε is the relative precision of analog circuitry.

Let $e_1 = x_1 - x^*$ and $e_2 = x_2 - x^*$. Then from the Hybrid algorithm,

$$r = -Ae_1, \tag{4}$$

so by (1),

$$y = -(A - E)^{-1} A e_1 + e.$$

Hence

$$e_2 = e_1 - (A - E)^{-1} A e_1 + e. \tag{5}$$

So,

$$\|I-(A-E)^{-1}A\| \leqq \frac{\varepsilon\kappa}{1-\varepsilon\kappa}, \tag{6}$$

where κ is the condition number of A in the $\|\cdot\|$ norm,

$$\kappa = \|A^{-1}\|\,\|A\|.$$

Furthermore,

$$\|(A-E)^{-1}A\| \leqq \frac{1}{1-\varepsilon\kappa}.$$

Combining this with (4) in (2), we get

$$\|e\| \leqq \frac{\varepsilon}{1-\varepsilon\kappa}\,\|e_1\|.$$

Using this estimate in (5), we obtain from (6) that

$$\|e_2\| \leqq \varepsilon\frac{\kappa+1}{1-\varepsilon\kappa}\,\|e_1\|. \tag{7}$$

This estimate furnishes a good error bound if

$$\varepsilon\kappa \ll 1. \tag{8}$$

Then we can expect each cycle of the algorithm to reduce the error by at least $\varepsilon\kappa$ (approximately).

From (7) we see that the effect of e on the error propagation is small compared to the effect of E. Then in (5) we formally drop e, viewing its effect to be absorbed into E. We shall see that the error matrix E is of the form

$$E = E_\varepsilon - \mu_0^{-1}I$$

for a simple analog model. Here E_ε is the error in analog representation of the entries of the matrix A and $\mu_0^{-1}I$ models the effect of finite amplification in the circuitry. Incorporating these observations into (5) gives

$$e_2 = Me_1,$$

where

$$M = I - [\mu_0^{-1}I + A - E_\varepsilon]^{-1}A. \tag{9}$$

Remark on sources of error. Analog computation will have three principal sources of error:

1) Digital-analog and analog-digital conversion of input and output (contained in (2)).

2) Digital-analog conversion error in representation of the matrix A (contained in (3)).

3) Effects of finite amplification and finite time (contributing to both (2) and (3)).

The third source of error will be analyzed in the next section.

In the following section we show that the time spent on the analog part of the computation is negligible in comparison to the digital part (if $\varepsilon\kappa \ll 1$). When A is symmetric, positive definite, the overrelaxed Jacobi method,

$$u \leftarrow u - \omega(Au - f),$$

has a convergence factor of

$$\frac{\kappa-1}{\kappa+1}$$

(when the optimal ω is used). When the hybrid and Jacobi methods are fully parallelized, the cost of one iteration of either method is approximately equal. The interesting cases in practice for the hybrid method are when both $\varepsilon\kappa \ll 1$ and $\kappa \gg 1$, so that the hybrid method is much faster than Jacobi.

Tables 1 and 2 contain the contraction factors (the error reduction per iteration factors) for the hybrid and Jacobi methods, respectively, for some sample values of ε and κ. The ratio of logarithms of contraction factors, $\log(\varepsilon\kappa)/\log((\kappa-1)/(\kappa+1))$, equals the ratio of the number of iterations required to attain a specified precision. Table 3 contains the ratios when the hybrid method converges, i.e., when $\varepsilon\kappa < 1$. The numbers represent speedup factors of the hybrid method over a comparable fully parallel (digital) simple iterative method, since an iteration of each method requires the same time. As the table demonstrates, the hybrid method can be more than a factor of 100 faster than optimally overrelaxed Jacobi. More sophisticated digital methods (e.g., conjugate gradients) are faster than Jacobi, but are not usually fully parallelizable.

A great deal of research in the past twenty years has been devoted to developing useful preconditioners for (digital) iterative methods (see [14]). Generally, an approximation to A is constructed which is close to A (in some sense), but much easier to factor. The corresponding preconditioned iterative method converges faster than the

TABLE 1
Contraction factors for the hybrid method.

$\varepsilon \setminus \kappa$	50	100	200	400
1/50	1.0000	2.0000	4.0000	8.0000
1/100	0.5000	1.0000	2.0000	4.0000
1/200	0.2500	0.5000	1.0000	2.0000
1/400	0.1250	0.2500	0.5000	1.0000
1/800	0.0625	0.1250	0.2500	0.50000

TABLE 2
Contraction factors for optimally overrelaxed Jacobi.

$- \setminus \kappa$	50	100	200	400
	0.9608	0.9802	0.9900	0.9950

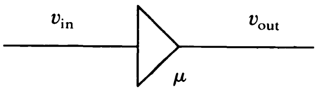
TABLE 3
Ratio of logarithms of contraction factors.

$\varepsilon \setminus \kappa$	50	100	200	400
1/50	-	-	-	-
1/100	17	-	-	-
1/200	35	35	-	-
1/400	52	69	69	-
1/800	69	104	139	139

original method, but costs more per iteration. These preconditioners typically reduce the error by considerably less than a digit per iteration. The analog solve step can be thought of as a preconditioning step, where the preconditioner is the original matrix A rounded off to nearly three digits. We can do this because we can prove that the analog step takes almost no time (in comparison to the digital steps). Table 1 demonstrates when we can expect to reduce the error by at least one digit per iteration.

3. A simple analog solver.

3.1. Basic model. The basic component of an analog circuit is an amplifier:



In the simplest approximation, the signals v_{in} and v_{out} satisfy the differential equation

$$\left(c\frac{d}{dt}+1\right)v_{\text{out}}=-\mu_0 v_{\text{in}}, \qquad t\geqq t_0, \tag{10}$$

where μ_0 is the steady-state gain, c is the time constant of the amplifier, and t_0 is the starting time. (Solid state amplifiers with $c<10^{-7}$ *s* and $\mu_0>10^4$ are currently available.)

Remark. In more general models, the differential operator $c(d/dt)+1$ in (10) is replaced by a polynomial in d/dt or, more generally, by an analytic function of d/dt, i.e., a pseudodifferential operator.

The transmission function μ of the amplifier is defined as

$$\mu=\mu(\omega)=\frac{v_{\text{out}}}{v_{\text{in}}}, \qquad v_{\text{in}}=e^{j\omega t},$$

where ω is the angular frequency and j is the imaginary unit. For (10),

$$\mu=\frac{-\mu_0}{1+cj\omega},$$

the so-called *one-pole* transmission function. (This relates our approach to conventional engineering terminology.)

Consider the network in Fig. 1. Here, the amplifier part consists of n identical amplifiers acting on n signals in parallel. Each amplifier is assumed to have the same transmission function $\mu=\mu(\omega)$ corresponding to a linear differential operator M:

$$\mu(\omega)=\frac{-\mu_0\,e^{j\omega t}}{M\,e^{j\omega t}}.$$

The output x of the amplifiers is processed by a passive network implementing multiplication by A, and then the residual $Ax-f$ is fed back to the input. In fact, this residual determination will be merged with the amplifiers into one circuit; Fig. 1 presents just a convenient equivalent model.

Remark. Note that the magnitude of the elements of A is limited since A cannot contain any amplification.

The state variable x satisfies the system of ordinary differential equations

$$Mx=-\mu_0(Ax-f).$$

In particular, for the one pole model (10), we have

$$\left(c\frac{d}{dt}+1\right)x=-\mu_0(Ax-f). \tag{11}$$

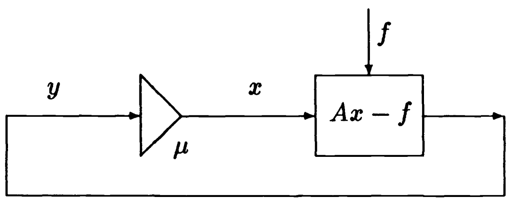
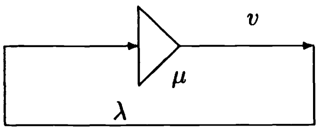


FIG. 1. *One component in a one-pole network.*

Now it is obvious that for $f=0$ and $Ax(t_0)=\lambda x(t_0)\neq 0$, we have

$$Mx=-\mu_0\lambda x, \qquad t\geqq t_0.$$

Let λ be an eigenvalue of A. We can thus reduce stability considerations to stability of the circuits of the form



For the one-pole model, we get the equation for the state variables

$$c\dot{v}+v=-\mu_0\lambda v,$$

where λ is an eigenvalue of A. Thus the one-pole model will be stable for all $\mu_0>0$ if and only if all real parts of the eigenvalues of A are positive.

3.2. Response speed and precision. Here we consider the one-pole model only. The state variable x satisfies the first-order system (11). To estimate the response, suppose that f is constant and that $x(0)=0$. Then (11) has the solution

$$x(t)=\bar{x}-\exp\left[-\frac{1}{c}(I+\mu_0A)t\right]\bar{x},$$

where $\bar{x}$ is the steady state,

$$\bar{x}=\left(\frac{1}{\mu_0}I+A\right)^{-1}f. \tag{12}$$

Now assume that A is symmetric, positive definite and let $\|\cdot\|$ be the l_2 norm. Then for the transient part of x, we have

$$\left\|\exp\left[-\frac{1}{c}(I+\mu_0A)t\right]\bar{x}\right\|\leqq e^{-1/c(1+\mu_0\lambda_{\min})t}\|\bar{x}\|.$$

The response time t_ε for relative precision ε is given by

$$-\frac{1}{c}(1+\mu_0\lambda_{\min})t_\varepsilon=\ln\varepsilon,$$

or

$$t_\varepsilon\approx\frac{c\ln(1/\varepsilon)}{\mu_0\lambda_{\min}}.$$

Since c is the time constant of the fast analog circuitry, meaningfully fast response is assured by the requirement $\mu_0\lambda_{\min}\gg 1$.

Example. Consider the typical values $\lambda_{\min} = 10^{-3}$, $\varepsilon = 10^{-4}$, $\mu = 10^5$, and $c = 10^{-7}$ s. Then

$$t_\varepsilon \approx \frac{10^{-7} \cdot 9}{10^5 \cdot 10^{-3}} s = 9 \cdot 10^{-9} s.$$

If $\|A\| \, \|A^{-1}\| \approx 10^3$ in the ℓ_∞ norm, then we get from (8) that the error reduction per iteration will be bounded approximately by

$$\varepsilon\kappa \approx 10^{-1}.$$

Conclusion. The total time required per iteration will be the sum of time for residual computation, digital-analog conversion time, a multiple of the time t_ε, analog-digital conversion time, and the time for addition. Because the time required for a digital operation can be assumed to be approximately c, the time constant of an analog circuit, we see that the time required for the analog solution itself is insignificant under the assumption that

$$\mu_0 \lambda_{\min} \gg 1. \tag{13}$$

However, we expect the contraction factor limitation (8), viz

$$\varepsilon\kappa \ll 1, \tag{14}$$

to be critical, because the attainable precision ε analog circuits is limited. The latter basically imposes a lower bound on $\lambda_{\min}$: assuming that

$$\kappa \approx \lambda_{\max} / \lambda_{\min}$$

and

$$\lambda_{\max} \approx 1,$$

we can write (14) as $\varepsilon / \lambda_{\min} \ll 1$, or

$$\varepsilon \ll \lambda_{\min}. \tag{15}$$

Note that we further need $\|I/\mu_0\| \leqq \varepsilon$ (cf. (12) and (3)). This condition is satisfied for typical parameters of contemporary devices. Thus, we require that

$$\mu_0 \varepsilon \gtrsim 1. \tag{16}$$

Note that (15) and (16) implies (13). Thus the time t_ε will never be significant when our estimates for the hybrid method are applicable.

3.3. Sample embodiment. We now consider a specific idealized circuit schema which embodies the analog part of the hybrid algorithm. We make use of classical devices: programmable resistors and operational amplifiers. The analog computational network will consist of n identical nodes as in Fig. 2. The resistors should be capable of attaining the values 0 and ∞. Each node has $n+1$ inputs $x_1, \cdots, x_n$ (the components of x) and f_i (one component of f). The output is x_i. It is connected to all inputs x_i of all nodes. In a practical implementation, most of the connections and most of the resistors will be missing. A fixed sparsity structure of A will be assumed. Such a sparsity structure may correspond to the discretization of a problem on a two-dimensional or three-dimensional mesh, or it might be a band structure.

The output x_i is given by the transmission function of the operational amplifier

$$x_i = \mu(v_+ - v_-).$$

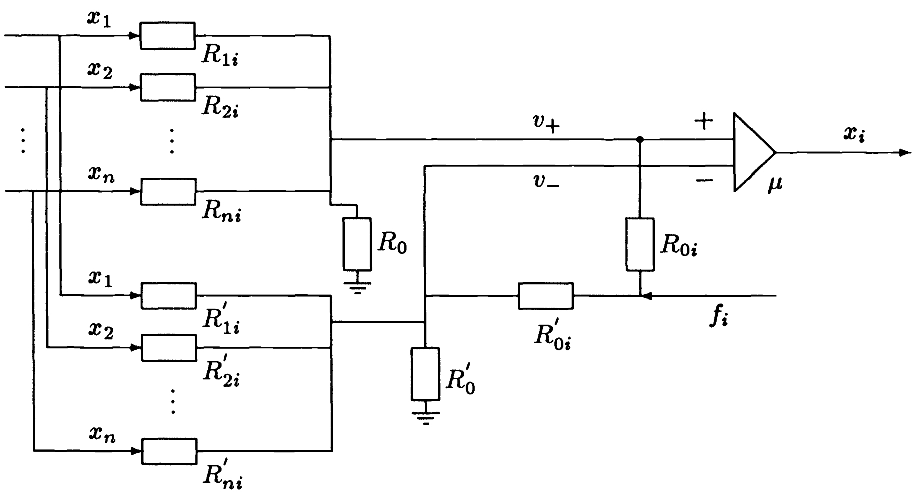


FIG. 2. *One-pole sample embodiment. All quantities* $x_1, \cdots, x_n, v_+, v_-, f_i$ *are voltages.*

Assuming zero output impedance and infinite input impedance of the operational amplifiers, the current balance at the inputs of the operational amplifier is

$$\frac{f_i - v_+}{R_{0i}} + \sum_j \frac{x_j - v_+}{R_{ji}} = \frac{v_+}{R_0}.$$

Then

$$v_+ = \frac{\sum_j \frac{x_j}{R_{ji}} + \frac{f_i}{R_{0i}}}{\frac{1}{R_0} + \sum_j \frac{1}{R_{ji}} + \frac{1}{R_{0i}}},$$

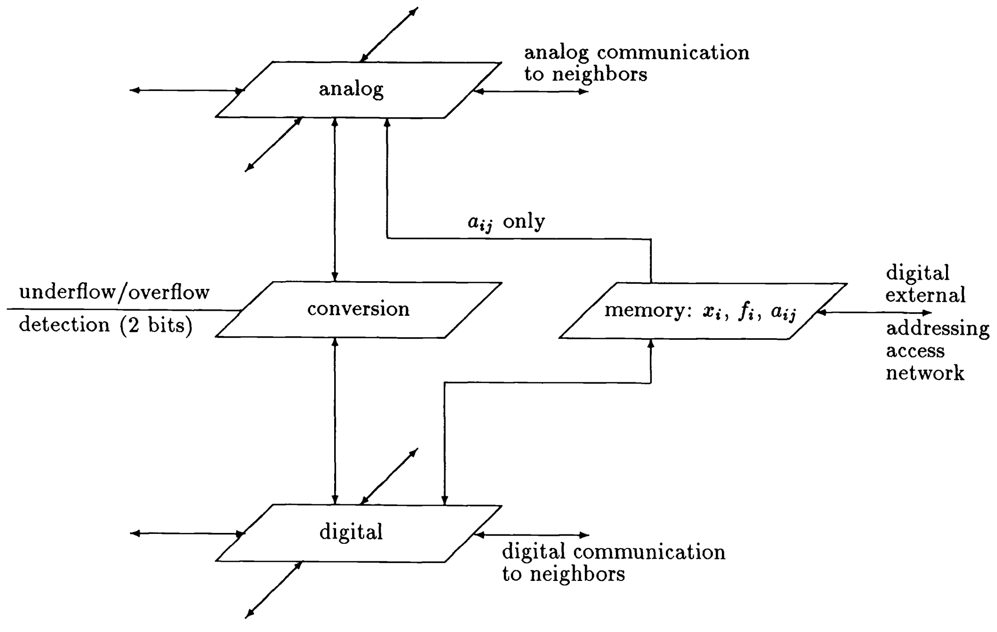

analog
analog communication to neighbors
a_{ij} only
underflow/overflow detection (2 bits)
conversion
memory: x_i, f_i, a_{ij}
digital external addressing access network
digital
digital communication to neighbors

FIG. 3. *Sample hybrid circuit for two-dimensional mesh geometry. This is part of a repeating chip pattern.*

and similarly for v_-. We can thus implement the transmission function of the node

$$x_i = -\mu\left(\sum_j a_{ij}x_j - f_i\right).$$

By expressing a_{ij} in terms of R_{ji} and R_0, and noting that resistances are nonnegative, we can show that

$$\sum_j |a_{ij}| < 2.$$

For a practical realization, a network using capacitors instead of resistors might be required. (Capacitors of high precision are easy to construct using the MOS 1C technique.)

An entire hybrid circuit using two-dimensional mesh geometry could look similar to the one in Fig. 3. An overflow/underflow detection two-line bus must be added to adjust the scaling of the residual fed into the analog solver. If the size of the analog output is too large, then the node which detects the overflow condition,

$$|x_i| > v_{\max},$$

will send a signal on bus line 1. Similarly, if a node detects the condition

$$|x_i| > v_{\min}, \qquad v_{\min} < v_{\max}/a, \qquad a > 1,$$

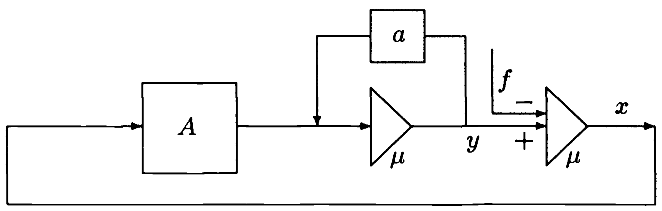
then it sends a signal on bus line 0. These bus lines are sensed by all conversion interface units. If bus line 1 is on, then all units decrease the analog right-hand side by the factor $1/a$, $a > 1$. If bus line 0 is off, then all units increase it by the factor a. This will guarantee that no $|x_i|$ is larger than $v_{\max}$ and at least one is larger than $v_{\min}$, thus making full use of available precision. The scaling factor is then used in the output analog/digital conversion and stored for the next iteration as a good initial guess.

This scaling can be easily implemented by a voltage multiplier/attenuator at the output of the conversion unit. An analog bus, using continuous adjustments of the scale, could also be considered.

4. A two-stage analog circuit.

4.1. Motivation. In the one-stage circuit, the scale of the output x and of the input f is, in general, different. Since $x \approx A^{-1}f$, x will be much larger than f. This can be a source of errors. Therefore, we consider an implementation of the product Ax using another amplifier. The right-hand side f is then easily combined with the output using a differential amplifier. The scaling of A also can be used to increase the speed of the circuit if necessary.

4.2. Basic model. Consider the network consisting of two amplifier arrays, feedback array a, and a passive network for Ax:



Assuming that all amplifiers have the same one pole transmission function

$$\mu = \frac{-\mu_0}{1 + cj\omega},$$

the constitutive equations of the network are

$$\begin{aligned} c\dot{y}+y&=-\mu_0(Ax+ay),\\ c\dot{x}+x&=-\mu_0(y+f). \end{aligned} \tag{17}$$

Here, x, y, and f consist of n parallel analog signals on separate lines. The feedback factor a is assumed to be the same for all components.

4.3. Analysis. Equations (17) have the steady-state solution (for constant f) given by

$$\bar{y}=-\mu_0(A\bar{x}+a\bar{y}), \qquad \bar{x}=-\mu_0(\bar{y}+f).$$

The first equation gives $\bar{y}=-\mu_0/(1+\mu_0 a)$. Then

$$\bar{x}=\left(\frac{1}{\mu_0}I+\frac{\mu_0 A}{1+\mu_0 a}\right)^{-1} f. \tag{18}$$

Thus $\lim_{\mu_0\to\infty}\bar{x}=aA^{-1}f$. The transient part of the solution is a solution of the homogeneous system

$$\begin{aligned} c\dot{y}&=-(1+\mu_0 a)y-\mu_0 Ax,\\ c\dot{x}&=\mu_0 y-x, \end{aligned} \tag{19}$$

with the initial condition being the error at $t=0$. Introducing the matrices

$$G=\begin{pmatrix} aI & A\\ -I & 0\end{pmatrix}, \qquad H=-\frac{1}{c}[I+\mu_0 G],$$

we can write (19) as

$$\begin{pmatrix}\dot{y}\\ \dot{x}\end{pmatrix}=H\begin{pmatrix}y\\ x\end{pmatrix}.$$

If λ is an eigenvalue of A, then G has two corresponding eigenvalues φ obtained as the eigenvalues of the matrix $\left(\begin{smallmatrix} a & \lambda\\ -1 & 0\end{smallmatrix}\right)$. In particular,

$$\varphi=\frac{a}{2}\pm\sqrt{\frac{a^2}{4}-\lambda}.$$

Thus the spectrum of H is

$$\sigma(H)=\left\{-\frac{1}{c}\left[1+\mu_0\left(\frac{a}{2}\pm\sqrt{\frac{a^2}{4}-\lambda}\right)\right]; \lambda\in\sigma(A)\right\}.$$

Suppose that A is symmetric and positive definite, and let $\lambda_{\min}$ denote its least eigenvalue. Then the element χ of $\sigma(H)$ with the largest real part (thus determining the response time) is equal to

$$\chi=-\frac{1}{c}\left[1+\mu_0\left(\frac{a}{2}-\sqrt{\frac{a^2}{4}-\lambda_{\min}}\right)\right].$$

The choice $a=2\lambda_{\min}$ gives the transient response (assuming $\lambda_{\min}\leqq 1$) for the slowest component

$$e^{\operatorname{Re}\chi t}=e^{-1/c(1+\mu_0\lambda_{\min})t}.$$

This is the same as for the one-stage circuit. However, by (18) the scaling of $\bar{x}$ is now

$$\bar{x}\approx aA^{-1}f=2\lambda_{\min}A^{-1}f.$$

Thus we can expect $\|x\| \approx 2\|f\|$. The response speed and precision analysis of § 3.2 holds for this case. The speed and precision are identical.

To speed up the transient response, we could use a larger feedback factor a. The choice

$$a = \sqrt{2\lambda_{\min}} \tag{20}$$

gives

$$-\operatorname{Re} \chi = \frac{1}{c}\left[1 + \mu_0 \frac{a}{2}\right] = \frac{1}{c}\left[1 + \mu_0 \sqrt{\frac{\lambda_{\min}}{2}}\right].$$

Using the technique from § 3.2

$$t_\varepsilon \approx \frac{c \ln \varepsilon}{\mu_0 \sqrt{\lambda_{\min}/2}}.$$

Meaningfully fast response is assured by the requirement $\mu_0\sqrt{\lambda_{\min}/2} \gg 1$.

Example. With the choice $a = \sqrt{2\lambda_{\min}}$ and considering the typical values $\lambda_{\min} = 10^{-3}$, $\varepsilon = 10^{-4}$, $\mu = 10^5$, and $c = 10^{-7}s$ once again, we have

$$t_\varepsilon \approx \frac{10^{-7} \cdot 9}{10^5 \cdot 22.4} s \approx 4 \cdot 10^{-11} s.$$

However, for $a = 2\lambda_{\min}$, we obtain $t_\varepsilon = 9 \cdot 10^{-9} s$. Thus choosing the feedback factor a according to (20) yields a significant speedup.

Remark. For the one-pole model of the amplifiers, the circuit is stable; however, compensation of the amplifiers so that they are well approximated by the one-pole model may be more critical here, because of both much stronger feedback and the presence of two amplifiers in the feedback loop.

4.4. Sample embodiment. An analog node will have inputs $x_1, \cdots, x_n, f_i$, and output x_i. In classical devices, such nodes were implemented using programmable resistors and two operational amplifiers (as in Fig. 4). Assuming infinite input and

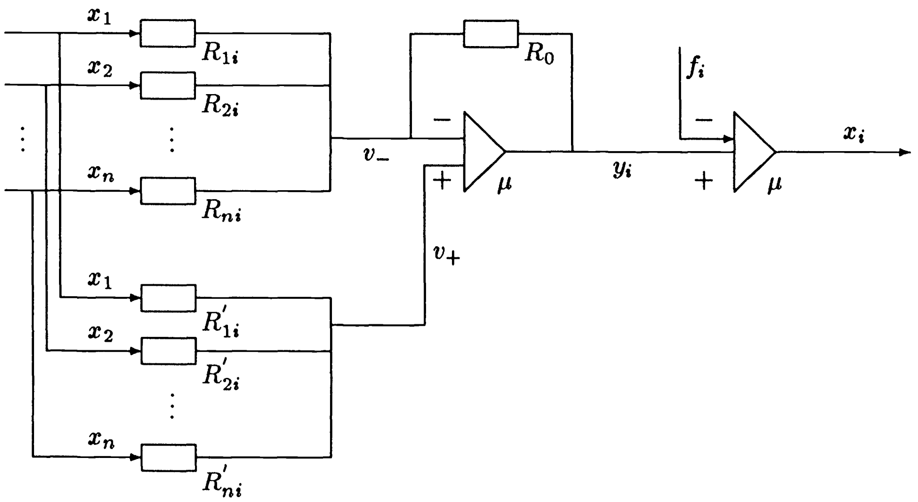


FIG. 4. *Two-stage sample embodiment. All quantities $x_1, \cdots, x_n, v_+, v_-, f_i$ are voltages. All resistors are programmable and can assume the values 0 and ∞.*

zero output impedances, Kirchhoff's laws yield

$$v_- = \frac{\sum_k \frac{x_k}{R_{ki}} + \frac{y_i}{R_0}}{\sum_k \frac{1}{R_{ki}} + \frac{1}{R_0}},$$

and

$$v_+ = \frac{\sum_k \frac{x_k}{R'_{ki}}}{\sum_k \frac{1}{R'_{ki}}}.$$

For the amplifiers, we have

$$y_i = \mu(v_+ - v_-), \qquad x_i = \mu(y_i - f_i).$$

Thus

$$y_i = \mu \left[\frac{\sum_k \frac{x_k}{R'_{ki}}}{\sum_k \frac{1}{R'_{ki}}} - \frac{\sum_k \frac{x_k}{R_{ki}} + \frac{y_i}{R_0}}{\sum_k \frac{1}{R_{ki}} + \frac{1}{R_0}} \right].$$

Then, setting

$$a = \frac{\frac{1}{R_0}}{\sum_k \frac{1}{R_{ki}} + \frac{1}{R_0}},$$

we have

$$y_i = \frac{\mu}{1 + a\mu} \sum a_{ik} x_k,$$

with

$$a_i k = \frac{\frac{1}{R'_{ki}}}{\sum_k \frac{1}{R'_{ki}}} - \frac{\frac{1}{R_{ki}}}{\sum_k \frac{1}{R_{ki}} + \frac{1}{R_0}}.$$

We see then that the hybrid realization here is similar to the one-stage solver.

5. A two-level algorithm. A more powerful version of the residual correction technique is the multilevel variant (see [2], [9]). In this section we consider the two-level version of the latter (see [4]), and we show how to apply the hybrid, analog/digital methods to it.

The two level method for solving the system $Ax = f$ is described as follows:

Step 1. Repeat p times: $x \leftarrow x - G(Ax - b)$
Step 2. $x \leftarrow x - PB^{-1}R(Ax - b)$.

Here the first step consists of p smoothing iterations using a scaled iterative procedure G (e.g., Jacobi, symmetric Gauss-Seidel, or conjugate gradients). The matrix R interpolates from the solution space onto a coarser space and P interpolates from the coarse space into the solution space. Typically, R is a linear interpolation method and P is R^T. A customary choice for B is $B = RAP$, with the dimension of B being considerably less than that of A.

Rewrite Step 2 as

$$\begin{aligned} r &= R(Ax-b), \\ y &= B^{-1}r, \\ x &\leftarrow x - Py. \end{aligned}$$

Our hybrid approach produces an analog solution $\bar{y}$ of $By = r$, where

$$\bar{y} = (B-E)^{-1}r, \qquad \|E\| \leqq \varepsilon \|B\|. \tag{21}$$

Note the error e in (1) is incorporated into E here, and E will vary from step to step. We assume that

$$\|(I - PB^{-1}RA)(I-GA)^p\| = a < 1, \tag{22}$$

a condition which the original two level method requires for convergence.

Now we study the effect of the limited precision (as characterized by ε) of the analog implementation of (21) which replaces y by $\bar{y}$.

Let e_1 be the error before step 1, e_2 the error after step 1, and e_3 the error after Step 2 in the two level method. Then

$$e_2 = (1-GA)^p e_1,$$

and

$$e_3 = (I - PB^{-1}RA)e_2.$$

The corresponding error $\bar{e}_3$ with the analog process invoked is

$$\bar{e}_3 = (I - P(B-E)^{-1}RA)e_2.$$

Assume that $\|z\| = \|Pz\|$ for all z and

$$\|(I-GA)^p\| \leqq C \tag{23}$$

(in most cases, $C = 1$). The following theorem quantifies the degree of degradation of the estimate (22) in the hybrid version.

THEOREM 1. *If* (22) *and* (23) *are satisfied, then*

$$\|(I - P(B-E)^{-1}RA)(I-GA)^p\| \leqq a + \frac{(C+a)\kappa(B)\varepsilon}{1-\kappa(B)\varepsilon},$$

or, equivalently,

$$\|\bar{e}_3\| \leqq \left[a + \frac{(C+a)\kappa(B)\varepsilon}{1-\kappa(B)\varepsilon}\right]\|e_1\|.$$

Proof. First consider $\bar{y} - y$. We have

$$\begin{aligned} \bar{y} - y &= [(B-E)^{-1} - B^{-1}]r \\ &= [(B-E)^{-1}B - I]y. \end{aligned}$$

Then (cf. (6)),

$$\|\bar{y} - y\| \leqq \frac{\kappa(B)\varepsilon}{1-\kappa(B)\varepsilon}\|y\|.$$

Now

$$\|y\| = \|Py\| = \|e_2 - e_3\| \leqq (C+a)\|e_1\|.$$

Hence

$$\|\bar{y} - y\| \leqq \frac{(C+a)\kappa(B)\varepsilon}{1-\kappa(B)\varepsilon}\|e_1\|.$$

Noting that

$$\bar{e}_3 = e_3 - P(\bar{y} - y),$$

we get

$$\|\bar{e}_3\| \leqq \|e_3\| + \|\bar{y} - y\| \leqq \left[a + \frac{(C+a)\kappa(B)\varepsilon}{1-\kappa(B)\varepsilon}\right]\|e_1\|. \qquad \square$$

6. Conclusions. We have analyzed a hybrid digital/analog algorithm and shown that it reduces the error per iteration by a considerable amount. In fact, most preconditioned iterative methods reduce the error by a fraction of this amount, moreover at a greater cost. The cost of the analog step has been shown to be negligible in comparison to that of the digital step. Furthermore, the cost of one iteration of the hybrid method is comparable to that of one iteration of a fully parallelized (digital) optimally overrelaxed Jacobi method. However, the hybrid method can be over 100 times faster than the corresponding Jacobi method for a fixed accuracy requirement. The technology exists now to build such a hybrid machine, either as a standalone computer or as a coprocessor board for a workstation.

REFERENCES

[1] M. A. G. Abushagur and H. J. Caulfield, *Bimodal optical computers*, in SPIE Proc. 939, 1988, pp. 29-33.

[2] R. E. Bank and C. C. Douglas, *Sharp estimates for multigrid rates of convergence with general smoothing and acceleration*, SIAM J. Numer. Anal., 22 (1985), pp. 617-633.

[3] A. Brandt, *Binary-input analog-computing* (BIAC) *hardware for numerical solution of elliptic problems*, Preliminary Report, Weizmann Institute of Sciences, Rehovot, Israel, 1969.

[4] F. Chatelin and W. L. Miranker, *Acceleration by aggregation of successive approximation methods*, J. Linear Algebra Appl., 24 (1980), pp. 17-47.

[5] W. K. Cheng and H. J. Caulfield, *Fully-parallel relaxation algebraic operations for optical computers*, Opt. Commun., 43 (1982), pp. 251-254.

[6] A. K. Ghosh, *Performance analysis of matrix preconditioning algorithms on parallel optical processors*, in SPIE Proc. 939, 1988, pp. 19-28.

[7] J. Jahns, *Concepts of optical digital computing-a survey*, Optik, 57 (1980), pp. 429-449.

[8] W. J. Karplus and R. A. Russell, *Increasing digital computer efficiency with the aid of error-correcting analog subroutines*, IEEE Trans. Comput., 20 (1971), pp. 831-837.

[9] J. Mandel, *Multigrid convergence for nonsymmetric, indefinite variational problems and one smoothing step*, Appl. Math. Comput., 19 (1986), pp. 201-216.

[10] J. Mandel and W. L. Miranker, *New techniques for fast hybrid solutions of systems of equations*, IBM Research Report RC 14157, Yorktown Heights, NY, 1988; Internat. J. Numer. Methods Engrg., to appear.

[11] D. Psaltis and R. A. Athale, *High accuracy computation with linear analog optical systems: A critical study*, Appl. Optim., 25 (1986), pp. 3071-3077.

[12] K. Watanabe and G. C. Temes, *A switched-capacitor digital multiplier*, in 1983 IEEE International Symposium on Circuits and Systems, Vol. 3, IEEE, Computer Society, New York, 1983, pp. 1270-1273.

[13] J. Wilkinson, *The Algebraic Eigenvalue Problem*, Oxford University Press, Oxford, UK, 1965.

[14] L. A. Hageman and D. M. Young, *Applied Iterative Analysis*, Academic Press, New York, 1981.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 6, pp. 1087-1101, November 1990

ITERATIVE SOLUTION OF A NONLINEAR SYSTEM ARISING IN PHASE-CHANGE PROBLEMS*

M. A. WILLIAMS† AND D. G. WILSON‡

Abstract. The relative merits of two iterative procedures for solving the mildly nonlinear equations arising from an implicit discretization of a three-dimensional Stefan problem were investigated. The two procedures are a selective successive overrelaxation (SOR) and a preconditioned conjugate gradient (PCG) method adapted specifically for use on this problem. (The SOR method for which numerical trials were run was actually Gauss-Seidel since, although the overrelaxation parameter was an input parameter, the value supplied was invariably one.) The algorithms were implemented in FORTRAN on an IBM 3090-200 S/VF and on a Cray X-MP 48. Vectorization was a major consideration. Although our variant PCG method converged in fewer iterations, the SOR iterations were shorter (in lines of code) and faster (in execution time). Thus, SOR required less time on both the IBM and Cray machines. Finally, SOR was found to be more robust than PCG.

Key words. iterative method, moving boundary, phase change, preconditioned conjugate gradient, Stefan problem, Gauss-Seidel

AMS(MOS) subject classifications. 65H, 65P, 80

1. Introduction.

The problem considered is a three-dimensional Stefan problem. This is a parabolic partial differential equation, with appropriate initial and boundary conditions, posed in a region that evolves with time. It represents a mathematical idealization of a phase-change process such as ice freezing. We consider a weak formulation called the "enthalpy" formulation in which a potentially troublesome internal boundary condition has been absorbed into the statement of the problem. The solution is a pair of functions $\{T, e\}$, where T and e denote temperature and enthalpy distributions, respectively. The location of the internal boundary between the two phases can be determined from these quantities.

The enthalpy formulation for phase-change problems and its implementation in numerical schemes has a long history. Enthalpy corresponds to internal energy content at constant pressure. The equations of the enthalpy formulation express conservation of energy, $\rho e_t - \operatorname{div} q = 0$ (where ρ is the density, e is the enthalpy, and q is the energy flux), movement of energy according to Fourier's law, $q = -k(T) \operatorname{grad} T$, and an equation of state relating energy and temperature. The first two of these are combined into equation (2.1) below, and the energy temperature relation is given in equations (2.2) and (2.3). Elliott and Ockendon [4] give a broad survey of moving boundary problems and practical methods, employing enthalpy formulations and their variants, for solving them.

Rose [8] defined an early explicit numerical enthalpy scheme in which the continuous temperature and the discontinuous enthalpy were both updated from timestep to timestep. Solomon [9] gave a modification of Rose's scheme in which the enthalpy

* Received by the editors September 12, 1988; accepted for publication (in revised form) September 5, 1989. This work was supported by the Applied Mathematical Sciences Subprogram of the Office of Energy Research, U.S. Department of Energy, and the Laboratory Graduate Participation Program of the Oak Ridge Associated Universities. Portions of this work appeared in [10].

† Department of Mathematics, Lehigh University, Bethlehem, Pennsylvania 18015. Present address, International Business Machines Corporation, Numerically Intensive Computing Center, Data Systems Division, Department 41UD, Mail Stop 276, Kingston, New York 12401.

‡ Mathematical Sciences Section, Engineering Physics and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831-8083.

was taken as the primary variable. (What he solved was $e_t = (k(e)e_x)_x$, where $k(e)$ is an adjusted thermal conductivity that has the correct value in the solid and liquid phases and that is small when the enthalpy is between zero and the latent heat of the phase transition. Although this scheme is difficult to justify from a physical point of view, it is exceptionally good numerically both in economy and in accuracy.) Athey [1] extended Rose's method, again updating both temperature and enthalpy at each timestep with an explicit difference scheme, to problems with internal heat sources. Implicit schemes for phase-change problems have evolved over several years. Meyer [6] presented and analyzed a finite-difference scheme, implicit in time, with a smoothed enthalpy. Jerome [5] analyzed another scheme using a backward Euler discretization in time without smoothing the enthalpy, and Elliott [2] presented yet another implicit scheme using finite elements. More recently, Elliott [3] has published an L^2 error bound, $O(h^{1/2})$, for this scheme.

In previous work we have used only explicit methods in the numerical solution of multidimensional moving boundary problems. For such schemes, the computations required at each time level are simple and can be performed quickly. However, stability considerations severely restrict the timestep size. More recently, we have adapted Elliott's ideas to finite-difference approximations for an enthalpy based model and implemented them in computer simultations of three-dimensional problems in Cartesian and cylindrical polar coordinates. The basic idea of the implicit scheme is that the enthalpy/temperature relation and the discrete approximation to the partial differential equation are solved simultaneously at the advanced timestep. Implicit methods are not subject to the stability restriction of the explicit scheme, and thus the number of time levels required to simulate a given problem may be reduced.

Implicit discretization of the governing equations leads to a mildly nonlinear system at each time level. Thus, while implicit methods require fewer time levels than explicit methods, the amount of computation per level is increased. We were pleasantly surprised to find that the implicit methods required less total computation time than the explicit method for a given simulation. Thus, for this problem, we have determined that the trade-off between larger timesteps at the expense of complicated computations and smaller timesteps with simpler computations favors the former. We have investigated iterative techniques for solving the nonlinear system, which may be written as a linear system in which the coefficient matrix depends on the solution vector. We have adapted the preconditioned conjugate gradient method for solving this specific nonlinear system, and our experience was that using this more sophisticated scheme for solving the nonlinear system was not worth the effort. We suspect that this is the case in general, but we have not investigated other such schemes.

In § 2 we state the multidimensional Stefan problem and give the numerical formulation. In § 3 we discuss a selective successive overrelaxation (SOR) algorithm suggested by Elliott [2], [4] for solving the resulting nonlinear system. In § 4 we describe a new adaptation of the preconditioned conjugate gradient (PCG) algorithm especially suited for phase-change problems. In § 5 we present results of numerical experiments with the SOR algorithm and the adapted PCG algorithm. In § 6 we summarize results and state our conclusions.

2. Problem formulation. The problem we consider is as follows. Determine a temperature distribution $T(\mathbf{x}, t)$ and an enthalpy distribution $e(\mathbf{x}, t)$ satisfying

$$(2.1)\quad \rho e_t = \operatorname{div}(k(T)\operatorname{grad} T) \quad \text{for } 0<x<l_x,\quad 0<y<l_y,\quad 0<z<l_z,\quad \text{and } t>0,$$

and

$$(2.2)\qquad e=\begin{cases} c_{\mathrm{Sol}}(T-T_{\mathrm{cr}}), & T<T_{\mathrm{cr}},\\ [0,H], & T=T_{\mathrm{cr}},\\ c_{\mathrm{Liq}}(T-T_{\mathrm{cr}})+H, & T>T_{\mathrm{cr}},\end{cases}$$

with appropriate initial and boundary conditions. Here e and T denote the enthalpy and temperature, respectively. T_{cr} and H denote the critical temperature and latent heat of the phase change, respectively. $k(T)$ denotes the thermal conductivity (which is different in solid and liquid phases and which could also be temperature-dependent, although we did not include this variation). c_{Sol} and c_{Liq} denote the constant specific heats of the solid and liquid, respectively. ρ denotes the constant density of the phase change material. $\mathbf{x}$ and t denote a position vector and time, respectively, and e_t denotes the partial derivative of enthalpy with respect to time. Note that because of the discontinuities involved, the partial derivatives of the temperature must be interpreted in a weak sense, and also that enthalpy is multivalued when $T=T_{\mathrm{cr}}$. Relation (2.2) can also be expressed as

$$(2.3)\qquad T=\begin{cases} T_{\mathrm{cr}}+\dfrac{e}{c_{\mathrm{Sol}}}, & e\leqq 0,\\ T_{\mathrm{cr}}, & 0<e<H,\\ T_{\mathrm{cr}}+\dfrac{e-H}{c_{\mathrm{Liq}}}, & e\geqq H,\end{cases}$$

where temperature is a single-valued function of enthalpy.

A mathematically equivalent formulation results if the Kirchoff transformation $u(T)=\int_{T_{\mathrm{cr}}}^{T} k(\tau)\,d\tau$ is applied. This transformation absorbs the material thermal conductivity into the definition of the temperature and thus avoids the troublesome problem of determining an equivalent thermal conductivity between cells of different materials. We present it here and discuss it briefly because Elliott's convergence proof for his SOR algorithm is for its application to this transformed problem, and also because its use simplifies somewhat the presentation of the PCG algorithm. For convenience we define $\beta(u)$ by

$$(2.4)\qquad \beta(u)=\begin{cases} \dfrac{u}{\gamma_{\mathrm{Liq}}}+\rho H & \text{for } u>0,\\ [0,\rho H] & \text{for } u=0,\\ \dfrac{u}{\gamma_{\mathrm{sol}}} & \text{for } u<0,\end{cases}$$

where $\gamma_{\mathrm{Liq}}\equiv k_{\mathrm{Liq}}/\rho c_{\mathrm{Liq}}$, and $\gamma_{\mathrm{Sol}}\equiv k_{\mathrm{Sol}}/\rho c_{\mathrm{Sol}}$. The transformed problem may be expressed as follows. Determine a "Kirchoff temperature" distribution $u(\mathbf{x},t)$ and a corresponding enthalpy distribution $E(\mathbf{x},t)$ satisfying

$$(2.5)\qquad E_t=\nabla^2 u \quad \text{for } 0<x<l_x,\quad 0<y<l_y,\quad 0<z<l_z,\quad \text{and } t>0,$$

and

$$E\in\beta(u),$$

with appropriate initial and boundary conditions.

For the discrete problem, we partition the domain $[0,l_x]\times[0,l_y]\times[0,l_z]$ into uniform, three-dimensional cells with nodes located at the centers of the cells. We do not require that the cells have the same dimensions in each direction, but each cell is identical to every other cell. Boundary nodes are located $\Delta x/2$, $\Delta y/2$, or $\Delta z/2$ away from any physical boundary. There are no nodes on the physical boundaries.

Let n_x, n_y, and n_z represent the numbers of cells (and hence nodes) in each of the x, y, and z directions, respectively. The dimensions of the cells are $\Delta x = l_x/n_x$, $\Delta y = l_y/n_y$, and $\Delta z = l_z/n_z$. (To make the computations with red/black ordering of the nodes convenient, we required n_x and n_y to be odd.) The node with subscripts ijk is an interior node if $1 < i < n_x$ and $1 < j < n_y$ and $1 < k < n_z$, otherwise it is a boundary node.

We discretize the partial differential equation (2.1) by replacing the time derivative by a standard forward difference quotient and spatial derivatives by central-like difference quotients. The spatial derivative terms are evaluated at time $n+\theta$ where $\theta \in [0,1]$ denotes the degree of implicitness of the scheme. By definition, $F^{n+\theta}$ *is* $(1-\theta)F^n + \theta F^{n+1}$. Substituting these difference quotients in (2.1) and rearranging terms gives the following discrete version at interior nodes:

$$\begin{aligned}
e^{n+1}_{ijk} = {} & \frac{\theta\Delta t}{\rho\Delta x}\left[k^{n+1}_{i+1/2,jk}\frac{(T^{n+1}_{i+1,jk}-T^{n+1}_{ijk})}{\Delta x} - k^{n+1}_{i-1/2,jk}\frac{(T^{n+1}_{ijk}-T^{n+1}_{i-1,jk})}{\Delta x}\right] \\
& + \frac{\theta\Delta t}{\rho\Delta y}\left[k^{n+1}_{i,j+1/2,k}\frac{(T^{n+1}_{i,j+1,k}-T^{n+1}_{ijk})}{\Delta y} - k^{n+1}_{i,j+1/2,k}\frac{(T^{n+1}_{ijk}-T^{n+1}_{i,j-1,k})}{\Delta y}\right] \\
& + \frac{\theta\Delta t}{\rho\Delta z}\left[k^{n+1}_{ij,k+1/2}\frac{(T^{n+1}_{ij,k+1}-T^{n+1}_{ijk})}{\Delta z} - k^{n+1}_{ij,k-1/2}\frac{(T^{n+1}_{ijk}-T^{n+1}_{ij,k-1})}{\Delta z}\right] + b^n_{ijk},
\end{aligned} \tag{2.6}$$

where

$$\begin{aligned}
b^n_{ijk} = e^n_{ijk} & + \frac{(1-\theta)\Delta t}{\rho\Delta x}\left[k^n_{i+1/2,jk}\frac{(T^n_{i+1,jk}-T^n_{ijk})}{\Delta x} - k^n_{i-1/2,jk}\frac{(T^n_{ijk}-T^n_{i-1,jk})}{\Delta x}\right] \\
& + \frac{(1-\theta)\Delta t}{\rho\Delta y}\left[k^n_{i,j+1/2,k}\frac{(T^n_{i,j+1,k}-T^n_{ijk})}{\Delta y} - k^n_{i,j-1/2,k}\frac{(T^n_{ijk}-T^n_{i,j-1,k})}{\Delta y}\right] \\
& + \frac{(1-\theta)\Delta t}{\rho\Delta z}\left[k^n_{ij,k+1/2}\frac{(T^n_{ij,k+1}+T^n_{ijk})}{\Delta z} - k^n_{ij,k-1/2}\frac{(T^n_{ijk}-T^n_{ij,k-1})}{\Delta z}\right].
\end{aligned} \tag{2.7}$$

The bracketed expressions in (2.6) and (2.7) may be interpreted as differences between heat fluxes into and out of the cell with subscripts ijk in the coordinate directions. At boundary cells where Neumann boundary conditions are given, the appropriate flux term is replaced by the specified boundary flux. At boundary cells where Dirichlet boundary conditions are given, the appropriate flux term is replaced by an equivalent flux calculated using the boundary temperature. See [11] and [12] for a more complete description of our treatment of boundary conditions.

The thermal conductivity between a node and one of its neighbors was taken to be the average of the thermal conductivities of the two corresponding cells. For example, $k_{i\pm1/2,jk}$ is defined by $k_{i\pm1/2,jk} = (k(T_{ijk}) + k(T_{i\pm1,jk}))/2$. If the contents of the cell with subscripts ijk and all its nearest neighbors are the same phase, then all thermal conductivities in (2.6) are equal and the spatial difference quotients reduce to standard central difference quotients.

With any ordering of the nodes, the difference equations may be written as

$$e^{n+1} + A(T^{n+1})T^{n+1} = b^n. \tag{2.8}$$

Here $A(T^{n+1})$ is an $N \times N$ element coefficient matrix and e^{n+1} and b^n are N element vectors where N is the total number of nodes ($N = n_x n_y n_z$).

The transformed problem (2.5) may be discretized similarly. The matrix equation is

$$E^{n+1}+Cu^{n+1}=f^n. \tag{2.9}$$

Here $E^{n_l}\in\beta(u^{n_l})$ for $l=1,\cdots,N$, C is a constant, symmetric, positive-definite, $N\times N$ matrix, and f^n is an N-vector. The matrix C is the negative of the finite-difference approximation to the Laplacian on the grid. The computational grid depends only on the spatial domain and is the same for both formulations of the problem.

When performing calculations, we assumed a red/black ordering of the nodes. In the red/black ordering, alternate nodes in the three-dimensional grid are identified as "red" and "black" as on a three-dimensional checkerboard. Thus all the neighbors of a given node of one color are of the opposite color. In the calculation, quantities associated with red and black nodes were updated alternately. Requiring the number of nodes in the x and y directions to be odd simplified the indexing and the vectorization of the loops implementing the calculation with red/black ordering. This is a coding detail that we need not discuss.

3. Elliott's selective SOR algorithm for phase-change problems. In this section, we describe an SOR algorithm to solve (2.8) for T^{n+1} and e^{n+1} simultaneously. Row l of the matrix equation (2.8) may be rewritten as

$$e_l^{n+1}+a_{ll}^{n+1}T_l^{n+1}=z_l^{n+1}, \tag{3.1}$$

where z_l^{n+1} is the sum of terms from time level n and terms of the form $a_{lj}T_j^{n+1}$ for $j\neq l$.

Let τ_l^p and η_l^p denote the pth approximations to T_l^{n+1} and e_l^{n+1}, respectively. Then the iterative form of (3.1) that we consider is

$$\eta_l^{p+1}+c_l^p\tau_l^{p+1}=z_l^{p^*}. \tag{3.2}$$

Starting with an initial approximation to T_l^{n+1}, τ_l^0, the quantity $z_l^{p^*}$ is computed using the most recently calculated approximations to $T_{\text{neighbor}}^{n+1}$ at neighboring nodes, i.e., $\tau_{\text{neighbor}}^{p+1}$ is used if it is available, otherwise $\tau_{\text{neighbor}}^{p}$ is used. The next Gauss-Seidel iterate $\tilde{\tau}_l^{p+1}$ is computed for successive p's from

$$\tilde{\tau}_l^{p+1}=\begin{cases}\dfrac{c_{\text{Sol}}T_{\text{cr}}+z_l^{p^*}}{c_l^p+c_{\text{Sol}}} & \text{if } z_l^{p^*}-c_l^pT_{\text{cr}}\leqq 0,\\ T_{cr} & \text{if } 0<z_l^{p^*}-c_l^pT_{\text{cr}}<H,\\ \dfrac{c_{\text{Liq}}T_{\text{cr}}+z_l^{p^*}-H}{c_l^p+c_{\text{Liq}}} & \text{if } H\leqq z_l^{p^*}-c_l^pT_{\text{cr}}.\end{cases} \tag{3.3}$$

This assignment results from solving the update equation (3.1) and a discrete version of the enthalpy temperature relation (2.3) for T^{n+1} and e^{n+1} simultaneously. It is explained in detail in both [10] and [13].

An updated Gauss-Seidel iterate for e^{n+1} is available when $\tilde{\tau}_l^{p+1}$ is computed. But, because the enthalpy values at the advanced timestep do not enter the right-hand side of the difference equations, it is not necessary to evaluate or record these successive approximations to e^{n+1}. The assignment of e^{n+1} can be deferred until the iteration for T^{n+1} has converged. Then the updated enthalpy is computed from the updated temperature using the discrete version of the partial differential equation. This ensures that the updated enthalpies and temperatures are self-consistent.

Equation (3.3) clearly shows the nonlinearity of the computation of the Gauss-Seidel iterates. In physical terms this nonlinearity results from (1) the enthalpy temperature relation being piecewise linear with different slopes in the solid and liquid phases, (2) the thermal conductivities between nodes (that enter the coefficients of the difference equations) being different in the solid and liquid phases, and (3) the location of the solid-liquid interface progressing through the domain of the problem. The appropriate mathematical statement is that the nonlinear update equations, consisting of the difference equations and the discrete enthalpy temperature relation, can be written (in a straightforward natural way) as a linear system in which both the elements of the coefficient matrix and the right-hand side depend on the solution vector.

A successive overrelaxation iterate is a linear combination of the previous iterate and the Gauss-Seidel iterate. The SOR iterate $\hat{\tau}$ is defined by

$$\hat{\tau}_l^{p+1} = \tau_l^p + \omega(\tilde{\tau}_l^{p+1} - \tau_l^p), \tag{3.4}$$

where $\omega \in (0, 2)$ is the relaxation parameter. In Elliott's selective SOR algorithm [2], [4] the Gauss-Seidel iterates are evaluated according to (3.3) and are used near the phase front, and the SOR iterates are used away from the phase front. More specifically, for each iteration and at each node the SOR temperature $\hat{\tau}_l^{p+1}$ is selected if $\hat{\tau}_l^{p+1}$ and τ_l^p lie on the same side of T_{cr}, and the Gauss-Seidel temperature $\tilde{\tau}_l^{p+1}$ is selected otherwise. Elliott recommended the tactic of using Gauss-Seidel near the phase front and SOR away from it to accelerate convergence away from the phase front and to avoid oscillations about the solid-liquid transition temperature near it. A convergence proof for this algorithm, applied to the transformed problem with the Kirchoff temperature, appears in [2]. Information on vectorized FORTRAN implementations, in Cartesian and cylindrical polar coordinates, for the Cray X-MP appears in [13].

In § 7.4 of [7], Ortega and Rheinboldt discuss generalized linear iterative techniques for multidimensional nonlinear problems. The basic idea is that an iterative method for solving a linear system, such as SOR, is combined with a one-dimensional iterative scheme for solving a nonlinear problem. By one-dimensional it is meant that there is only one unknown. As an example consider the one-dimensional Newton's method for finding a zero of one function of a single real variable $f(x) = 0$ by the iteration $x_{n+1} = x_n - f(x_n)/f'(x_n)$. The extension to multiple dimensions is $x_{n+1} = x_n + \Delta_n$, where now x and Δ are N element vectors, f is a vector-valued function of a vector variable, and Δ_n is the solution of the linear system $Jf(x_n)\Delta = -f(x_n)$, where $Jf(x_n)$ is the $N \times N$ element Jacobian of f evaluated at the previous iterate. The solution of this linear system could be obtained by an iterative scheme, so that the computation of the vector x would be done as a combination of inner and outer iterations. Alternatively, the computation of the solution vector could be done, as in Elliott's scheme, by the direct application of the principle of the iterative scheme for the linear system in the nonlinear setting. Ortega and Rheinboldt [7] say that a general characteristic of such methods is that, except under special circumstances, they exhibit only a linear rate of convergence. They give several references, dating back to 1963, on the use and analysis of schemes employing SOR as the iterative method for solving the linear system.

4. A PCG algorithm adapted for phase-change problems. In this section we describe a preconditioned conjugate gradient algorithm (PCGA) adapted for solving the nonlinear system (2.9). It can be shown (see [10]) that the solution of this system minimizes the nondifferentiable convex functional given by

$$J(w) = \frac{1}{2} w^T C w - w^T f^n + \sum_{j=1}^{N} \phi(w_j), \tag{4.1}$$

where $\phi(\cdot)$ is a piecewise quadratic function defined by

$$\phi(s)=\begin{cases}\dfrac{1}{2\gamma}s^2+\rho Hs & \text{for } s>0,\\ \dfrac{1}{2\gamma}s^2 & \text{for } s\leqq 0.\end{cases} \tag{4.2}$$

Observe that $\partial\phi(u)$, the subgradient of ϕ, is just $\beta(u)$, (2.4). (If Φ is differentiable at u, then $\partial\Phi(u)$ is the derivative of Φ at u. If Φ is not differentiable at u, then $\partial\Phi(u)$ is an upper bound on difference quotients from the left and a lower bound on difference quotients from the right.) The minimum of a differentiable convex functional occurs at a point where its gradient is zero. The minimum of a nondifferentiable convex functional occurs at a point where zero is an element of its subgradient.

In both the solid region and the liquid region, $E(\cdot)$ is linear in u. The nonlinearity at $u=0$ is due solely to the change in enthalpy associated with the latent heat. We rewrite the vector E^{n+1} as $D(u^{n+1})u^{n+1}+L(u^{n+1})$, where $D(u^{n+1})=(d_u)$ is an $N\times N$ diagonal matrix and $L(u^{n+1})=(l_l)$ is an N element vector. D and L are defined as follows:

$$d_{ll}(u^{n+1})=\begin{cases}\gamma_{\text{Sol}}^{-1}, & u_l^{n+1}<0,\\ 0, & u_l^{n+1}=0,\\ \gamma_{\text{Liq}}^{-1}, & u_l^{n+1}>0,\end{cases} \quad \text{and}$$

$$l_l=\begin{cases}0, & u_l^{n+1}<0,\\ E^{n+1}, & u_l^{n+1}=0,\\ \rho H, & u_l^{n+1}>0.\end{cases} \tag{4.3}$$

In physical terms, $D(u^{n+1})u^{n+1}$ represents sensible heat and $L(u^{n+1})$ represents latent heat. Using this notation, the nonlinear system (2.9) can be rewritten as

$$(C+D(u^{n+1}))u^{n+1}=f^n-L(u^{n+1}). \tag{4.4}$$

The jump discontinuity at $u=0$ is absorbed in the definition of $L(u^{n+1})$. Defining $A(u^{n+1})\equiv C+D(u^{n+1})$ and $b(u^{n+1})\equiv f^n-L(u^{n+1})$, the system (4.4) can be written as $A(u^{n+1})u^{n+1}=b(u^{n+1})$.

Writing the system in this way suggests a nested iteration. The outer iteration is on $A(u^{n+1})$ and $b(u^{n+1})$, and the inner iteration is on u^{n+1} and E^{n+1}. Let μ^p and η^p denote the pth iterative approximations to u^{n+1} and E^{n+1}, respectively. Then, given initial approximations μ^0 and η^0, successive iterates are calculated as follows.

```
for $p=0,1,2,\cdots$
    construct $A(\mu^p)$ and $b(\mu^p)$;
    if $\|b(\mu^p)-A(\mu^p)\mu^p\|\leqq\varepsilon$, then stop;
    else
      compute $\mu^{p+1}$ by applying one step of an iterative
          method to $A(\mu^p)x=b(\mu^p)$.
      compute $\eta^{p+1}$ using $\eta^{p+1}=f^n-C\mu^{p+1}$.
```

Clearly, for each μ^p the matrix $A(\mu^p)$ is symmetric and positive definite. Thus, a PCG algorithm can be used in the inner iteration. We think of the nested iteration procedure with a PCG algorithm in the inner iteration as a variant preconditioned conjugate gradient algorithm adapted for use on our system of mildly nonlinear equations. We present the PCGA algorithm here deferring momentarily a discussion of the preconditioning.

```
Given $\mu^0$ and $\eta^0$,
For $p = 0, 1, 2, \cdots$
    $r^p = b(\mu^p) - A(\mu^p)\mu^p$;
    if $\|r^p\| \leqq \varepsilon$,
    then set $u^{n+1} = \mu^p$,
        set $E^{n+1} = \eta^p$, and stop;
    else
      if $p = 0$,
      then $s^1 = r^0$;
      else
        solve $M^p z^p = r^p$,
        $\beta_p = -\dfrac{(s^p)^T A(\mu^p) z^p}{(s^p)^T A(\mu^p) s^p}$,
        $s^{p+1} = z^p + \beta_p s^p$;
        $\alpha^{p+1} = \dfrac{(s^{p+1})^T z^p}{(s^{p+1})^T A(\mu^p) s^{p+1}}$,
        $\mu^{p+1} = \mu^p + \alpha^{p+1} s^{p+1}$,
        $\eta^{p+1} = f^n - C\mu^{p+1}$.
```

M^p is a preconditioning matrix. The superscript indicates that M may change between iterations to reflect changes in $A(\mu^p)$. Choices of preconditioner are discussed in the next section.

In the PCG algorithm for linear systems, the search directions are A-orthogonal. Choosing β_p according to the PCGA algorithm guarantees that the search directions are "A-orthogonal" with the latest "A." More precisely, after p iterations of the PCGA algorithm, $(s^i)^T A(\mu^i) s^{i+1} = 0$ for $i = 0, 1, \cdots, p-1$.

Because we felt it would be more physically realistic, we used the discrete version of the original formulation (2.1) with distinct thermal conductivities, instead of the discrete version of the transformed problem (2.5) with the thermal conductivities absorbed into the Kirchoff temperature. In the discrete version of the original formulation, the coefficient matrix cannot be represented as the sum of a constant matrix and a diagonal matrix. However, the discussion remains valid, if in equation (4.4) (and in the algorithm definitions that follow it) the constant coefficient matrix C is replaced by $C(u)$. Thus, in the experiments we did, the PCGA algorithm was applied as stated with minor changes in the computation of η^{p+1} to take into account the different thermal conductivities in the different phases (as explained in the second paragraph after (2.7)).

5. Results of numerical experiments. In this section, we compare the performance of the PCGA algorithm with that of the SOR algorithm discussed in § 3. We have investigated the PCGA algorithm with diagonal preconditioning and with one-step symmetric successive overrelaxation (SSOR) preconditioning. SSOR preconditioning was more effective for the class of problems we considered. The SSOR preconditioning step consists of a forward and backward, that is, a red-black-red, sweep of Elliott's SOR algorithm. The forward and backward sweeps of the SSOR are necessary to maintain symmetry of the preconditioner matrix. Since the SSOR preconditioner was more effective than a diagonal preconditioner, we present only the results of numerical experiments with the PCGA algorithm with one-step SSOR preconditioning. We have used these methods on numerous test problems with various initial and boundary conditions. The data presented here are representative results.

We considered a one-phase Stefan problem with the following initial and boundary data:

Box dimensions: $1.2\ \mathrm{m} \times 1.0\ \mathrm{m} \times 0.8\ \mathrm{m}$.

Initial conditions: $T(x, y, z, 0) = 0°\mathrm{C}$, $e(x, y, z, 0) = 15.0\ \mathrm{kJ/kg}$.

Boundary conditions: $q(x, y, z, t) = \partial T/\partial n(x, y, z, t) = 0\ \mathrm{kJ/m^3}$ on the sides, $x = 0$, l_x and $y = 0$, l_y and top, $z = l_z$; $T(x, y) = -10 - 3x - 4y^2$ on the bottom, $z = 0$.

We assumed the following thermal and physical properties:

Physical constants: Critical temperature: $T_{\mathrm{cr}} = 0°\mathrm{C}$.
Thermal conductivity
of solid: $k_{\mathrm{Sol}} = 1.0 \times 10^{-3}\ \mathrm{kJ/m\text{-}s\text{-}°C}$
of liquid: $k_{\mathrm{Liq}} = 1.0 \times 10^{-2}\ \mathrm{kJ/m\text{-}s\text{-}°C}$
Specific heat: $c_{\mathrm{Sol}} = c_{\mathrm{Liq}} = 1.0\ \mathrm{kJ/kg\text{-}°C}$
Density: $\rho = 1.0\ \mathrm{kg/m^3}$
Latent heat: $H = 15.0\ \mathrm{kJ/kg}$.

It must be conceded that these thermophysical property values do not correspond to any real material. They were chosen to create a representative process that would progress rapidly, make the computations interesting, and reduce the cost of the numerical experiments.

The Stefan number St is defined by $\mathrm{St} = c_{\mathrm{Sol}}(T_{\mathrm{rep}} - T_{\mathrm{cr}})/H$. Here T_{rep} is a representative temperature in the problem. The Stefan number associated with the thermal properties listed above is about 1.0. When the Stefan number is small ($\leqq 0.1$), there are analytic techniques that accurately approximate the solution. When the Stefan number is large ($\geqq 10$), the problem resembles that of heat transfer without change of phase. We have successfully simulated phase-change problems with Stefan numbers of different orders of magnitude. However, we present results for only the most interesting case, $\mathrm{St} \approx 1$.

The basic parameters for the numerical methods are the number of nodes in each direction, nx, ny, and nz, the timestep size Δt, the degree of implicitness θ, and the tolerance ε, which defines the stopping criterion for the inner iterations. We compare the performance of the PCGA and SOR algorithms as a function of each of the last three parameters. Unless otherwise stated, the parameters have the following values: $nx = 21$, $ny = 15$, $nz = 45$, $\Delta t = 10 \times$ maximum stable explicit timestep size, $\theta = \frac{1}{2}$, and $\varepsilon = 10^{-4}$. The maximum stable explicit timestep size is easily computed as a function of the thermophysical parameters and the cell dimensions. It is the minimum over all possible combinations of heat capacities and thermal conductivities of the Courant numbers

$$\rho c_{\mathrm{Sol\,or\,Liq}} \left\{ \left(\frac{k_{\mathrm{Sol\,or\,Liq}}}{\Delta x^2} + \frac{k_{\mathrm{Sol\,or\,Liq}}}{\Delta y^2} + \frac{k_{\mathrm{Sol\,or\,Liq}}}{\Delta z^2} \right) \right\}^{-1}.$$

The relaxation parameter ω was taken to be 1.0 in both the SOR algorithm and the SSOR preconditioner (thus reducing SOR to Gauss–Seidel). Other choices of ω probably would accelerate the convergence of the SOR algorithm. Other choices of ω might also accelerate the convergence of the PCGA algorithm with SSOR preconditioning, although we are less optimistic about this. However, since the coefficient matrix is recomputed at each iteration of the PCGA and is changing dynamically throughout each SOR iteration, the determination of an appropriate (not to mention "optimal") value of ω for either algorithm is a task that we have not attempted. (Elliott and

Ockendon [4] suggested computing a value of ω at each node at each timestep as a weighted average of the optimal values of ω for the two linear heat transfer problems involving only the single liquid and solid phases where the weights correspond to the relative amounts of solid and liquid present at the most recent time. But taking ω equal to 1.0 seemed good enough for our purpose.)

Qualitatively, the SOR algorithm appears to be more robust than PCGA. The latter has failed to converge for timesteps larger than 40 times the maximum stable explicit timestep size. For these cases, SOR required a large number of iterations but did finally converge. We also did a few numerical experiments with the algorithms on two-phase Stefan problems (for which the initial temperature is not the critical temperature). For these problems, the PCGA algorithm sometimes suffered from oscillations about T_{cr}, while Elliott's selective SOR scheme did not.

Quantitative comparisons of the algorithms are based on computation time and number of iterations required for convergence. Both algorithms were implemented in FORTRAN on an IBM 3090 and also on a Cray X-MP. Except for the form of the compiler directives, and the specification of double precision for the IBM version (which resulted in the same precision as the Cray version), the code was exactly the same for both machines. It was not our purpose to compare performance of the machines, but rather that of the algorithms. The processing times were, of course, different on the two machines, but the relative performance of the algorithms on each was comparable. (One machine is somewhat faster than the other, but the slower machine has more memory available and can thus accommodate larger problems. Because we used a cell-centered finite-volume discretization, a refinement of a three-dimensional problem results in an increase in the number of cells by a factor of 27, and powers of 27 grow rapidly.) Computation times given are the actual CPU times, in seconds, for one processor of a dedicated IBM 3090-200 S/VF.

5.1. Performance as a function of timestep size. The advantage of implicit methods (with $\theta \geqq \frac{1}{2}$) over explicit methods is that the timestep size is not restricted by stability considerations. However, since implicit methods require solving a nonlinear system at each time level, they require more computation time per timestep than explicit methods. If the cost of solving the nonlinear system can be offset by taking large timesteps, then implicit methods can be effective. If not, then explicit methods may be more effective. Thus the performance of the nonlinear system solver as a function of timestep size is important.

Figure 1 shows the average number of iterations required for convergence for various timestep sizes for the SOR and PCGA algorithms. Timestep sizes are shown as multiples of the maximum stable timestep for the explicit discretization. The maximum stable explicit timestep size for the problem described above is $\Delta t = 3.5 \times 10^{-3}$. Each run simulated 1.5 seconds of the phase-change process. (By this time the box was about 13 percent solidified.) The PCGA algorithm always required fewer iterations for convergence.

For both methods, the initial approximation was taken to be the result of an explicit update with the maximum stable explicit timestep size instead of the distribution from the previous timestep. The motivation for this was that the computation was nearly free, since the terms corresponding to an explicit update were required anyway, and the result of an explicit update was felt to be a step in the right direction of the implicit update with a larger timestep size. We would expect that as the ratio of the implicit timestep size to the maximum stable explicit timestep size increased, the average number of iterations required for convergence would increase. Figure 1 shows

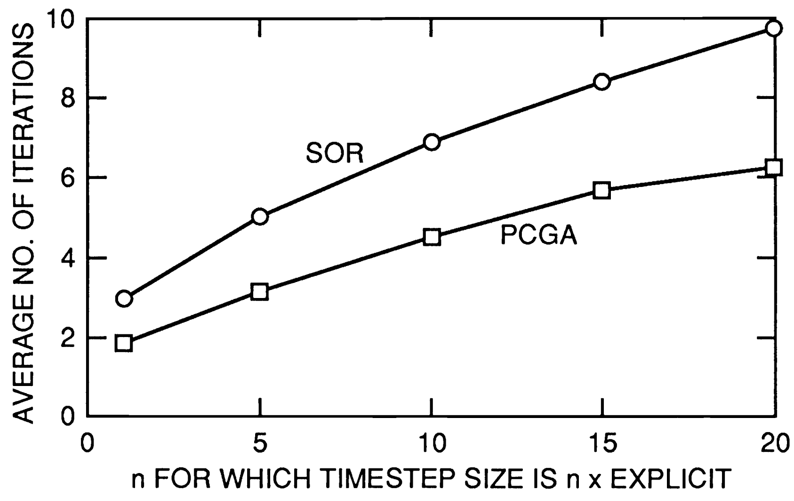


FIG. 1. *Average number of iterations versus timestep size.*

that this effect is more pronounced for SOR than for PCGA. (Since the implicit and explicit difference equations are not the same, more than one iteration is required even when the implicit and explicit timestep sizes are equal.)

A PCGA iteration is more time consuming than an SOR iteration. Thus, even though PCGA required fewer iterations for convergence, Fig. 2 shows that SOR required less execution time. For both algorithms, the total computation time decreases sharply as the timestep size increases since fewer steps are needed to reach the final time. Simulating 1.5 seconds of the phase-change process using an explicit method required 29.3 seconds of IBM 3090 S/VF execution time. For timesteps equal to 10, 15, and 20 times the maximum stable explicit timestep size, the implicit method with the SOR solver required less execution time than the explicit method. For time steps equal to 15 and 20 times the maximum stable explicit timestep size, the implicit method with the PCGA solver required less execution time than the explicit method. Thus, implicit methods can be cost effective for such problems.

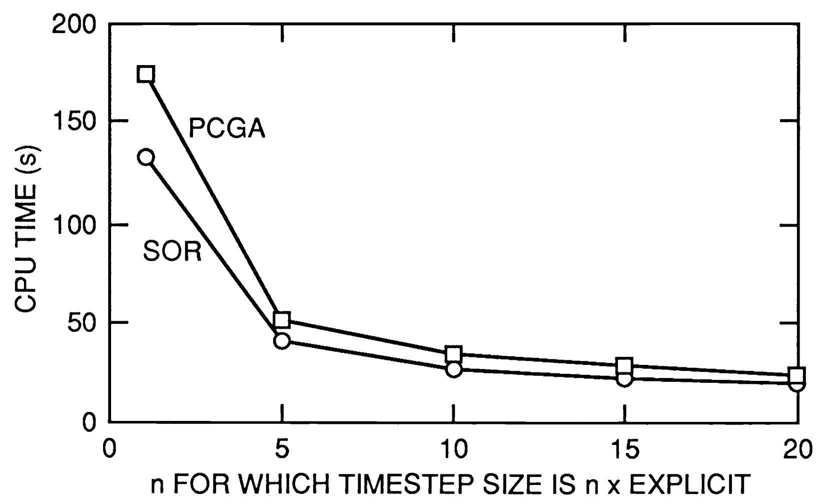


FIG. 2. CPU *time versus timestep size.*

5.2. Performance as a function of degree of implicitness. As discussed in § 3, the degree of implicitness may vary between zero and one. When $\theta = 0$, the method is fully explicit. When $\theta = \frac{1}{2}$, the method is a Crank-Nicolson discretization. When $\theta = 1$, the method is fully implicit. Choosing $\theta = \frac{1}{2}$ gives a higher order truncation error in regions away from the phase front. We would expect the number of iterations required for convergence to increase as θ increases. In Table 1 we present performance results for three representative values of θ. For stability reasons, we considered only values of $\theta \geqq \frac{1}{2}$. As expected, the average number of iterations increased as θ increased. The variation is nearly linear for both methods, with the slope of the PCGA line slightly greater than the slope of the SOR line. Again, the PCGA algorithm required fewer iterations but more total CPU time than the SOR algorithm.

5.3. Performance as a function of tolerance. Figure 3 shows how the average number of iterations varies with the tolerance, ε. Both the SOR curve and the PCGA curve are almost linear. The slope of the SOR line is about 1.0, corresponding to a linear rate of convergence. The slope of the PCGA line is about 0.8, corresponding to a superlinear rate of convergence. However, for reasonable values of the convergence tolerance, SOR required less total execution time than PCGA. (For values of $\varepsilon < 10^{-10}$, PCGA required less execution time than SOR.)

5.4. Performance as a function of problem size. An operational method of verifying the convergence of a discrete solution to a partial differential equation is to generate

TABLE 1
Performance (on an IBM 3090-200S/VF) *as a function of* θ.

	SOR		PCGA	
θ	Average number of iterations	CPU Time (sec)	Average number of iterations	CPU Time (sec)
0.5	6.9	93.8	4.5	115.8
0.75	8.0	106.8	5.2	132.6
1.0	9.1	120.9	6.0	149.2

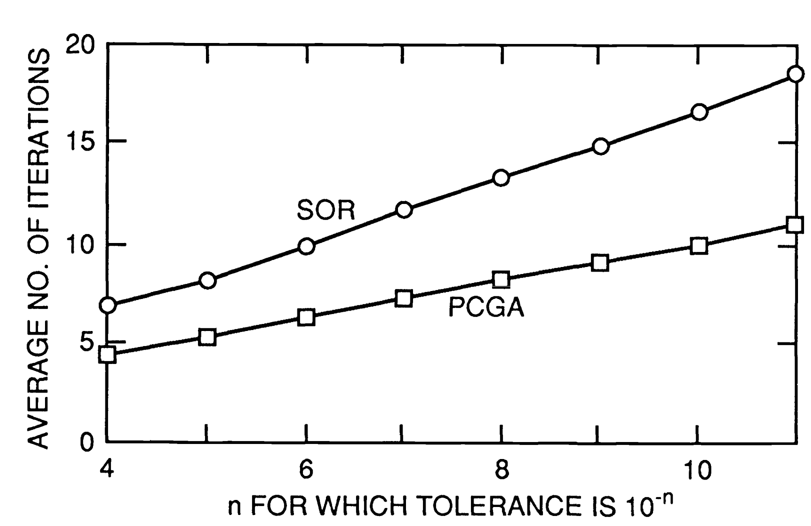


FIG. 3. *Average number of iterations versus tolerance.*

output on several refinements of the mesh and compare results. Good agreement inspires confidence. Thus larger and larger versions of a problem may be run successively. It is important to know if the superior performance of the SOR algorithm is degraded as larger and larger problems are considered. Figure 4 shows a comparison of the performance of the SOR and PCGA algorithms on a sequence of problems obtained by simulated refinements of a basic mesh.

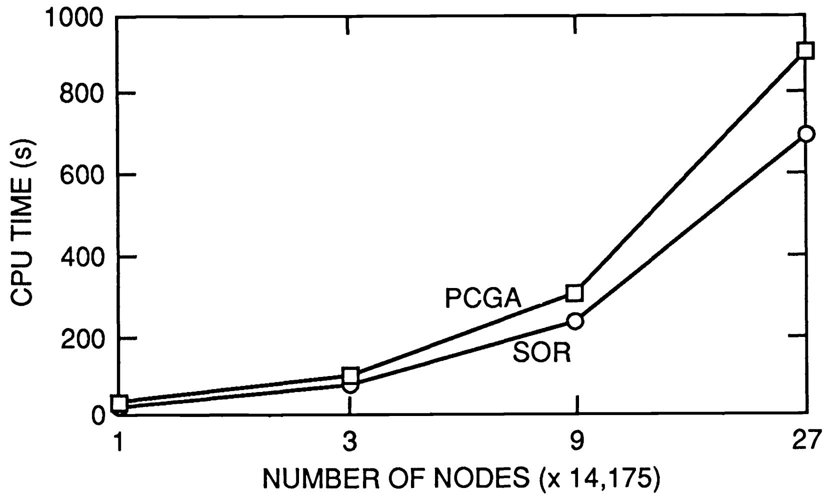


FIG. 4. CPU *time versus problem size.*

As stated earlier, we use a cell-centered finite-volume discretization. Thus to retain the same nodes in a refined mesh the number of nodes must grow either by factors of three or by factors that contain three. To generate the data shown in Fig. 4, we considered a base problem with 14,175 nodes ($21 \times 15 \times 45$). Our purpose was to compare algorithms instead of to verify results. In addition, we wanted to keep the timestep size constant (at 10 times the maximum stable explicit timestep size). Therefore, we simulated successive refinements by considering larger problems with the same meshsizes in each direction, but with more nodes. In the second problem, the number of cells in the x direction was increased by a factor of three, resulting in 42,525 nodes ($63 \times 15 \times 45$). In the third and fourth problems, the number of cells in each of the y and z directions was also increased by a factor of three, resulting in 127,575 ($63 \times 45 \times 45$) and 382,725 ($63 \times 45 \times 135$) nodes, respectively. (One repetition of this pseudorefinement would have resulted in a problem with more than ten million nodes.)

Figure 4 shows that the SOR algorithm required less computing time than the PCGA algorithm for each of these problems, and furthermore the disparity is increasing in favor of the SOR algorithm as the problem size increases. We have tried both algorithms on larger and smaller problems, including successive three-dimensional refinements of somewhat smaller problems. The results presented are representative.

6. Summary and conclusion. There are many questions associated with the use of PCG for solving a nonlinear system. It was not our goal to investigate these questions, but to determine if implicit methods could be cost effective for multidimensional moving boundary problems and to identify a good technique for solving the resulting mildly nonlinear equations. Based on the results presented here, we conclude that implicit methods can be used effectively for modeling phase-change problems. However, we are pessimistic about the usefulness of techniques more sophisticated than SOR for solving the nonlinear equations.

The appeal of the preconditioned conjugate gradient algorithm for solving linear systems is that it converges in far fewer iterations than other iterative methods. When SOR or Jacobi's method requires many iterations for convergence, PCG requires less total computing time. This frequently occurs when a good initial approximation for the solution is not available. In the norm used to test for convergence, the PCG method is not monotonically convergent and thus there is no guarantee that one step of the iteration will improve an initial approximation to the solution. Therefore, it is remarkable that the nonlinear, one-step, PCGA algorithm defined in § 4 converges. One could argue that several inner iterations should be taken until the norm of the residual has been reduced to a specified level. But this tactic results in spending iterations computing a more accurate approximation to the solution of the wrong problem.

The PCGA algorithm we have developed is effective for solving the nonlinear system generated by an implicit discretization of a one-phase Stefan problem. It converges in fewer iterations than Elliott's SOR algorithm described in § 3 and it exhibits a superlinear rate of convergence. However, an SOR iteration requires less computation than a PCGA iteration. If SOR required a large number of iterations to converge, PCGA might be computationally more efficient. But, because good initial approximations are readily available, we observed that SOR rarely required more than 15-20 iterations to converge. Thus, SOR required less execution time, on both the IBM and Cray machines, for the problems we considered and we expect this to be the general case.

Phase-change problems involve moving discontinuities in the coefficients, one of the solution functions and the gradient of the other, and this causes the coefficient matrix to change frequently. In Elliott's selective SOR algorithm, a matrix element is computed when it is needed. The computations involve the most recently computed values and the coefficient matrix is continually changing during each iteration. In the PCGA algorithm, matrix elements are updated in an outer iteration and they remain fixed during the inner iteration. Thus it seems that the SOR algorithm can deal with the discontinuities more efficiently.

Acknowledgments. The paper has benefited from a critical reading by Charles Romine and John Drake of the Mathematical Sciences Section. Most of their numerous suggestions for improvement have been incorporated into the paper, as have several others from two kindly anonymous referees.

REFERENCES

[1] D. R. ATHEY, *A finite difference scheme for melting problems*, J. Inst. Math. Appl., 13 (1974), pp. 353-366.

[2] C. M. ELLIOTT, *On the finite element approximation of an elliptic variational inequality arising from an implicit time discretization of the Stefan problem*, IMA J. Numer. Anal., 1 (1981), pp. 115-125.

[3] ———, *Error analysis of the enthalpy method for the Stefan problem*, IMA J. Numer. Anal., 7 (1987), pp. 61-71.

[4] C. M. ELLIOTT AND J. R. OCKENDON, *Weak and Variational Methods for Moving Boundary Problems*, Pitman, London, 1982.

[5] J. W. JEROME, *Nonlinear equations of evolution and a generalized Stefan problem*, J. Differential Equations, 26 (1977), pp. 240-2261.

[6] G. H. MEYER, *Multidimensional Stefan problems*, SIAM J. Numer. Anal., 10 (1973), pp. 522-538.

[7] J. M. ORTEGA AND W. C. RHEINBOLDT, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970.

[8] M. E. ROSE, *A method for calculating solutions of parabolic equations with a free boundary*, Math. Comp., 14 (1960), pp. 249-256.

[9] A. SOLOMON, *Some remarks on the Stefan problem*, Math. Comp., 20 (1966), pp. 347–360.

[10] M. A. WILLIAMS, *Iterative solution of a nonlinear system arising in phase change problems*, Ph.D. thesis, Lehigh University, Bethlehem, PA, 1987; Report ORNL-6398, Oak Ridge National Laboratory, Oak Ridge, TN, 1987.

[11] M. A. WILLIAMS AND D. G. WILSON, *Vectorized difference schemes for a three dimensional enthalpy formulation for phase change problems*, Report ORNL/TM-10034, Oak Ridge National Laboratory, Oak Ridge, TN, 1986.

[12] ———, IMPSOR, *a fully vectorized* FORTRAN *code for three dimensional moving boundary problems with Dirichlet or Neumann boundary conditions*, Report ORNL-6393, Oak Ridge National Laboratory, Oak Ridge, TN, 1987.

[13] D. G. WILSON AND R. E. FLANERY, *Modeling cyclic melting and refreezing in a hollow metal canister*, Report ORNL-6497, Oak Ridge National Laboratory, Oak Ridge, TN, 1988.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 6, pp. 1102-1115, November 1990

ALGORITHMS FOR THE POLAR DECOMPOSITION*

WALTER GANDER†

Abstract. For the polar decomposition of a square nonsingular matrix, Higham [*SIAM J. Sci. Statist. Comput.*, 7 (1986), pp. 1160-1174] has given a reliable quadratically convergent algorithm that is based on Newton's iteration. Motivated by Halley's iteration, the author constructs a new family of methods that contains both methods (Higham's and Halley's) as special cases. These methods generalize to rectangular matrices and some of them are also useful in computing the polar decomposition of rank deficient matrices.

Key words. polar decomposition, singular value decomposition, Higham's iteration, Halley's iteration, rank deficient matrices

AMS(MOS) subject classifications. 65F20, 65F30, 65F99

1. Introduction. The polar decomposition of an $m\times n$ matrix A can be defined (and computed) using the singular value decomposition (SVD) [7]. Let $A=U\Sigma V^T$ be the SVD (with U and V orthogonal and Σ diagonal). If $m\geqq n$, then the decomposition exists with the $m\times n$ matrix U and the $n\times n$ matrices Σ and V. Inserting $V^TV=I_n$ before Σ, we obtain the polar factors Q and M

$$A=UV^TV\Sigma V^T=QM \tag{1.1}$$

where $Q=UV^T$ is an orthogonal $m\times n$ matrix and $M=V\Sigma V^T$ is $n\times n$ symmetric and positive (semi-) definite. If $m<n$ the SVD of A is obtained by transposing the SVD of A^T. The dimension of the matrices are then $m\times m$ for U, $m\times m$ for Σ, and $n\times m$ for V. Again, we insert $V^TV=I_m$ before Σ and obtain (1.1). Here the $m\times n$ matrix $Q=UV^T$ has orthogonal rows: $QQ^T=I_m$ and the $n\times n$ matrix $M=V\Sigma V^T=QM$ has at least $n-m$ zero eigenvalues. If A is rank deficient the decomposition is not unique [5].

The polar decomposition is being used in new algorithms, which are designed for new computer architectures. Some of these algorithms characterized as *rich in matrix-vector multiplication* [1] are given in [9] to compute *block reflectors*, generalizations of Householder transformations.

In [4] Higham proposes an iterative algorithm to compute the orthogonal polar factor of a nonsingular $n\times n$ matrix A. This algorithm is based on the well-known Newton-iteration to compute the square root of a number. Starting with $X_0=A$, in Higham's algorithm the sequence X_k is computed by the iteration

$$X_{k+1}=\frac{1}{2}\left(\gamma_kX_k+\frac{1}{\gamma_k}X_k^{-T}\right), \tag{1.2}$$

where γ_k is an acceleration parameter. The present paper was motivated by using Halley's iteration [6] to compute the square root. The equivalent iteration

$$X_{k+1}=X_k(X_k^TX_k+3I)(3X_k^TX_k+I)^{-1} \tag{1.3}$$

is more general than (1.2), since it also works for rectangular matrices!

Earlier Björck and Bowie [2] showed that the iteration starting with $A_0=A$,

$$A_{k+1}=A_k\left(I+\frac{1}{2}T_k+\frac{3}{8}T_k^2+\cdots+(-1)^p\binom{-1/2}{p}T_k^p\right)$$

* Received by the editors January 30, 1989; accepted for publication September 1, 1989.

† Institut für Wissenschaftliches Rechnen, ETH Zentrum, CH-8092 Zürich (na.gander@na-net.stanford.edu).

and $T_k = I - A_k^T A_k$ converges to Q with order $p+1$. Again this algorithm works for rectangular matrices; however, the sequence converges only if $\|I - A^T A\| < c_p$ for some constants c_p (see [2]).

A simple change in Higham's algorithm, factoring X_k out of the bracket, gives

$$X_{k+1} = \frac{1}{2} X_k \left(\gamma_k I + \frac{1}{\gamma_k} (X_k^T X_k)^{-1} \right), \tag{1.4}$$

an algorithm that works again also for rectangular matrices. In this paper we propose a new family of iteration methods that contains Higham's, Halley's, and Björck and Bowie's algorithm (for $p = 1$) as a special case. These methods converge globally. It is well known that by forming $A^T A$ numerically one can lose information in A. Since all the new methods (also Björck and Bowie's algorithm) use the product $X_k^T X_k$, they should be used only for rectangular or singular matrices, where Higham's method fails.

In [5] an algorithm to compute the polar decomposition of an arbitrary matrix is proposed. The idea is to *use Higham's method for square matrices and to apply it to the triangular matrix obtained from a complete orthogonal decomposition of the original matrix.* For this decomposition one has to take a rank decision and use classical numerical software. The algorithms proposed here need no rank decision and no initial transformation. They use, like Higham's method for square matrices, only matrix operations that vectorize and parallelize well.

All the computations were done using MATLAB [8] on a SUN workstation. The machine precision is $\varepsilon = 2.2204e-16$.

2. Analysis of the iteration. We consider the following iteration:

$$\begin{aligned} X_0 &= A, \\ X_{k+1} &= X_k h(X_k^T X_k) = h(X_k X_k^T) X_k, \end{aligned} \tag{2.5}$$

where the function h is to be specified. Let A be a $m \times n$ matrix with $m \geqq n$ and let $A = U\binom{\Sigma_0}{0} V^T$ be its singular value decomposition.

LEMMA 2.1. *The iteration* (2.5) *is equivalent to*

$$\begin{aligned} \Sigma_1 &= \Sigma_0 \\ \Sigma_{k+1} &= \Sigma_k h(\Sigma_k^2) = h(\Sigma_k^2)\Sigma_k \end{aligned} \tag{2.6}$$

and

$$X_k = U \begin{pmatrix} \Sigma_k \\ 0 \end{pmatrix} V^T.$$

The proof of this lemma by induction is straightforward and omitted.

We wish to choose h so that $\Sigma_k \to I$ as fast as possible. Then

$$\lim_{k \to \infty} X_k = UV^T = Q.$$

Note that (2.6) represents n uncoupled scalar iterations:

$$x_{k+1} = x_k h(x_k^2) \quad \text{where } x_0 = \sigma_i, \qquad i = 1, \cdots, n. \tag{2.7}$$

It is well known that the iteration

$$x_{k+1} = F(x_k) = x_k h(x_k^2) \tag{2.8}$$

converges with order p to the fixed point s if and only if $s = F(s)$ and

$$F'(s) = F''(s) = \cdots F^{(p-1)}(s) = 0, \qquad F^{(p)}(s) \neq 0. \tag{2.9}$$

In our case the fixed point we wish to reach is $s=1$. This implies that $h(1)=1$. Note, however, that if $h(1)=1$ then also $s=0$ and $s=-1$ are fixed points of the iteration (2.8). This means that for $\sigma_i=0$ because of (2.7), the iteration (2.5) will not converge to the desired limit. Methods of the type (2.5) are therefore theoretically restricted to matrices A with full rank. *However, as we will see in the sequel, the rounding errors usually make these methods work also for rank deficient matrices.*

Equations (2.9) lead to the following conditions for h:

$$\begin{aligned} F &= xh(x^2), \\ F' &= h(x^2)+2x^2h'(x^2), \\ F'' &= 6xh'(x^2)+4x^3h''(x^2), \\ F''' &= 6h'(x^2)+24x^2h''(x^2)+8x^4h'''(x^2), \\ F^{(4)} &= 60xh''(x^2)+80x^3h'''(x^2)+16x^5h^{(4)}(x^2). \end{aligned} \tag{2.10}$$

For the fixed point $s=1$ we obtain

$$\begin{aligned} F(1) &= 1 \Rightarrow h(1) = 1, \\ F'(1) &= 0 \Rightarrow h'(1) = -\tfrac{1}{2}, \\ F''(1) &= 0 \Rightarrow h''(1) = \tfrac{3}{4}, \\ F'''(1) &= 0 \Rightarrow h'''(1) = -\tfrac{15}{8}, \\ F^{(4)}(1) &= 0 \Rightarrow h^{(4)}(1) = \tfrac{105}{16}. \end{aligned} \tag{2.11}$$

The function h therefore approximates a function w whose Taylor series is

$$w(1+t)=1-\frac{1}{2}t+\frac{3}{4\cdot 2!}t^2-\frac{15}{8\cdot 3!}t^3+\frac{105}{16\cdot 4!}t^4\mp\cdots$$

and we recognize that it is

$$w(1+t)=\frac{1}{\sqrt{1+t}}=1-\frac{1}{2}t+\frac{3}{8}t^2-\frac{5}{16}t^3+\frac{35}{128}t^4\mp\cdots. \tag{2.12}$$

THEOREM 1. *The iteration* (2.5) *is of order* $p+1$ *if and only if the function* h *approximates* $w(x)=1/\sqrt{x}$ *such that*

$$h^{(i)}(1)=w^{(i)}(1) \quad \textit{for } i=0,1,\cdots,p.$$

Examples.

(1) Algorithms of Björck and Bowie. Here h is a polynomial, the partial sum of the expansion (2.12) including the term t^p. Therefore the method is of order $p+1$. Especially for $p=1$ we have the second order method

$$h(x^2)=1-\frac{1}{2}(x^2-1).$$

(2) Higham's (nonaccelerated) algorithm. Here we have

$$X_{k+1}=\frac{1}{2}(X_k+X_k^{-T})=\frac{1}{2}X_k(I+(X_k^TX_k)^{-1}) \tag{2.13}$$

$$=\frac{1}{2}(I+(X_kX_k^T)^{-1})X_k. \tag{2.14}$$

Therefore $h(x^2)=\frac{1}{2}(1+1/x^2)$ and, putting $x^2=1+t$, we get the expansion

$$h(1+t)=\frac{1}{2}\left(1+\frac{1}{1+t}\right)=1-\frac{1}{2}t+\frac{1}{2}t^2\mp\cdots,$$

which shows that the method has the order 2.

(3) Halley's iteration. Here $h(x^2)=(x^2+3)/(3x^2+1)$. Putting $x^2=1+t$ and comparing the expansion

$$h(1+t)=\frac{4+t}{3t+4}=1-\frac{1}{2}t+\frac{3}{8}t^2-\frac{9}{32}t^3\pm\cdots$$

with (2.12) we see that this method is of order 3.

3. Higham's generalized algorithm for rectangular matrices. The speed of convergence of the nonaccelerated version of Higham's algorithm (2.13) may be initially very slow, since the error is reduced only roughly by a factor of 2. In [4] an acceleration factor $\gamma^{(k)}$ therefore is introduced:

$$X_{k+1}=\frac{1}{2}\left(\gamma^{(k)}X_k+\frac{1}{\gamma^{(k)}}X_k^{-T}\right).$$

Higham computes an optimal value for $\gamma^{(k)}$ minimizing an error bound:

$$\gamma_{\text{opt}}^{(k)}=\frac{1}{\sqrt{\sigma_1(X_k)\sigma_n(X_k)}}. \tag{3.15}$$

Let $\sigma_1,\cdots,\sigma_n$ be the singular values and $\kappa:=\sigma_1/\sigma_n$ the condition number of X_k. Using the optimal value (3.15), the matrix X_k is scaled with the geometric mean $\gamma=1/\sqrt{\sigma_1\sigma_n}$. The scaled matrix $1/\sqrt{\sigma_1\sigma_n}\,X_k$ has its singular values σ_i' distributed equally on both sides of 1 ($\sqrt{\kappa}=\sigma_1'\geqq\cdots\geqq\sigma_n'=1/\sqrt{\kappa}$), which is the best one can do for the next iteration step.

Now let A be $m\times n$ with full rank. We consider the iteration $X_0=A$ and

$$\begin{aligned} X_{k+1}&=\frac{1}{2}X_k\left(\gamma^{(k)}I+\frac{1}{\gamma^{(k)}}(X_k^TX_k)^{-1}\right)\\ &=\frac{1}{2}\left(\gamma^{(k)}I+\frac{1}{\gamma^{(k)}}(X_kX_k^T)^{-1}\right)X_k, \end{aligned} \tag{3.16}$$

where one has to use the first or the second form according to whether $m\geqq n$ ($X_k^TX_k$ is $n\times n$ nonsingular) or $m<n$ (then $X_kX_k^T$ is $m\times m$ nonsingular).

An estimation for (3.15) can be obtained from the matrix $B=X_k^TX_k$, respectively, $B=X_kX_k^T$. Using

$$\gamma_{\text{est}}^{(k)}=\sqrt[4]{\frac{\|B^{-1}\|_1}{\|B\|_1}}, \tag{3.17}$$

it is not difficult to see that $\gamma_{\text{opt}}^{(k)}/\sqrt[8]{n}\leqq\gamma_{\text{est}}^{(k)}\leqq\sqrt[8]{n}\,\gamma_{\text{opt}}^{(k)}$.

Example. For $A=\text{rand}\,(20,10)$ a matrix with random elements and $\kappa(A)=15.5$, we computed the polar decomposition of A and A^T. The results were obtained in seven iterations and are given in Table 1.

TABLE 1

$\|UH-A\|_2/\|A\|_2=2.6838e-16$	$\|U^TU-I\|_2=2.3015e-16$
$\|UH-A^T\|_2/\|A^T\|_2=1.7818e-16$	$\|UU^T-I\|_2=1.2199e-16$

4. A family of iteration methods. Higham's generalized algorithm works well and reliably if the matrix A is not too ill-conditioned. The aim of this section is to obtain an algorithm that also works well when A is rank deficient *without having to determine in advance the numerical rank.*

To show important ideas, we consider as an example the *gallery*(5) matrix from MATLAB [8]. This matrix is (5×5) and has rank 4. Using Halley's iteration, we obtain the decomposition $A=UH$ with $\|U^TU-I\|_2=5.1932e-16$ and $\|UH-A\|_2/\|A\|_2=6.2770e-8$ in 14 iterations. The results are not quite satisfactory. The reason is that rounding errors perturb the iterates because the matrices $C_k=I+3X_k^TX_k$, especially the first one, are ill-conditioned ($\kappa(C_0)=3.0634e+10$). The condition of the following matrices C_k improves more and more since the singular values of X_k converge to 1.

We can improve the condition of C_k by simply scaling the matrix A. Using the above algorithm with $X_0:=A/\|A\|_2$ yields the decomposition $A=UH$ with $\|U^TU-I\|_2=3.6020e-16$ and $\|UH-A\|_2/\|A\|_2=5.2335e-16$ in 39 iterations. Now $\kappa(C_0)=4$ and the results are as accurate as one can wish. However, the computational effort with so many iterations is too large. However, it is important to note that the number of iterations *does not depend on the size of the matrix, since all the singular values are increased simultaneously*!

The matrix considered in this example has one zero singular value. Theoretically this is a fixed point of Halley's iteration. Since this fixed point is not attractive and since, due to rounding errors, the singular value does not remain exactly zero during the iterations, it converges in finite arithmetic to 1, to the desired limit.

Though Halley's iteration converges cubically, global convergence is poor. It is not difficult to see that the global error is roughly reduced by a factor of 3, which can be too slow for practical purposes.

Let $A=QM$ be an $m\times n$ matrix and h a real rational function with

$$h(t)>0 \quad \text{for} \quad t>0. \tag{4.18}$$

Then

$$\begin{aligned} B&=Ah(A^TA)=h(AA^T)A\\ &=QMh(M^2)=Qh(M^2)M. \end{aligned} \tag{4.19}$$

Let $M=VDV^T$ with $D=\operatorname{diag}\{\lambda_i\}\lambda_i\geqq 0$ be the eigenvalues of M. Then because of (4.18)

$$\begin{aligned} Mh(M^2)&=VDV^TVh(D^2)V^T\\ &=VDh(D^2)V^T=\text{pos. (semi-) def.} \end{aligned}$$

and therefore (4.19) is the polar decomposition of B with $M_B=Mh(M^2)=h(M^2)M$.

We now consider iteration functions F with the rational function

$$h(x^2)=\frac{c+x^2}{d+fx^2}.$$

We want to determine the two parameters c and d such that we have quadratic convergence, i.e., from the equations $h(1)=1$ and $h'(1)=-\frac{1}{2}$. A short calculation yields the condition $d\neq -f$, i.e., $f\neq 1$ and $d=f-2$ and $c=2f-3$. For every value of the parameter f with $f\neq 1$, the iteration function

$$F(x)=xh(x^2)=x\frac{2f-3+x^2}{f-2+fx^2} \tag{4.20}$$

leads to a quadratically convergent iteration method.

THEOREM 2. *Let A be an $m \times n$ matrix with full rank and polar decomposition $A = QM$. With suitably chosen scaling parameter α, the iteration $X_0 = \alpha A$*

$$\begin{aligned} X_{k+1} &= X_k((2f-3)I + X_k^T X_k)((f-2)I + fX_k^T X_k)^{-1}, \quad m \geqq n \\ &= ((2f-3)I + X_k X_k^T)((f-2)I + fX_k X_k^T)^{-1} X_k, \quad m \leqq n \end{aligned} \tag{4.21}$$

converges quadratically to Q for every $f \neq 1$.

Note the following special cases of iteration (4.21): for $f = 0$ we obtain Björck and Bowie's algorithm (for $p = 1$). $f = 2$ yields Higham's and $f = 3$ yields Halley's iteration.

Proof. By construction of (4.20) local quadratic convergence is assured. We have to show that for every value of f the iteration also converges globally. We discuss the iteration function (4.20) for positive x and various values of f. Let $\sigma_1 \geqq \sigma_2 \geqq \cdots \geqq \sigma_n > 0$ be the singular values of X_k and similarly let σ_i' be the singular values of X_{k+1}.

(1) $f > 3$ (see Fig. 1, e.g., for $f = 100$): Large singular values $\sigma_i \gg 1$ are reduced by about the factor f per iteration while small $\sigma_i < 1$ increase but stay smaller than one. F has a minimum at

$$x_e = \sqrt{\frac{(f-2)(2f-3)}{f}} \tag{4.22}$$

and

$$F(x_e) = \left(\sqrt{\frac{2f-3}{f}}\right)^3 \frac{1}{\sqrt{f-2}} \sim 2\sqrt{\frac{2}{f}}.$$

Therefore we have the following bounds for the new σ_i' corresponding to $\sigma_i > 1$:

$$0 < F(x_e) \leqq \sigma_i' \leqq F(\sigma_1),$$

which is approximately

$$2\sqrt{\frac{2}{f}} \leqq \sigma_i' \leqq \frac{\sigma_1}{f}.$$

We have in this case global convergence for any $\alpha > 0$.

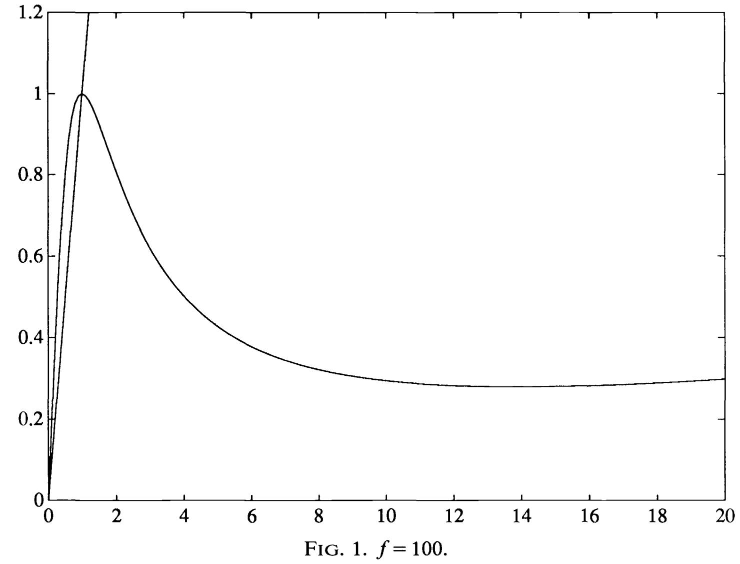


FIG. 1. $f = 100$.

(2) $f=3$ Halley's iteration (see Fig. 2). $F(x)$ is monotonically increasing. Large $\sigma_i>1$ are decreased roughly by a factor of 3 and small $\sigma_i<1$ are increased roughly by the same factor. Also here for any $\alpha>0$ we have global convergence.

(3) $2<f<3$ (see Fig. 3, e.g., for $f=2.1$). Here we have $0<x_e<1$ and x_e is a local maximum. For values of f close to 2, say $f=2+\varepsilon$, we have

$$F(x_e)=\frac{1}{\sqrt{\varepsilon}}\left(\sqrt{\frac{1+2\varepsilon}{2+\varepsilon}}\right)^3\sim\frac{1}{2}\frac{1}{\sqrt{2\varepsilon}} \tag{4.23}$$

and

$$F'(0)=\frac{2f-3}{f-2}=\frac{1+2\varepsilon}{\varepsilon}\sim\frac{1}{\varepsilon}.$$

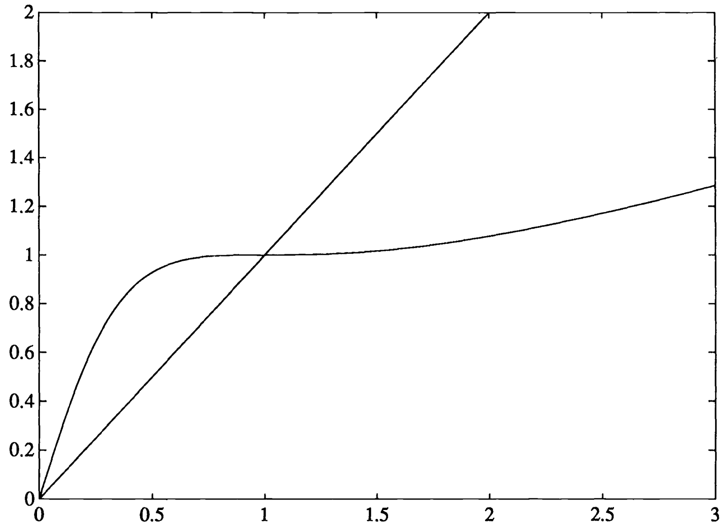


FIG. 2. $f=3$.

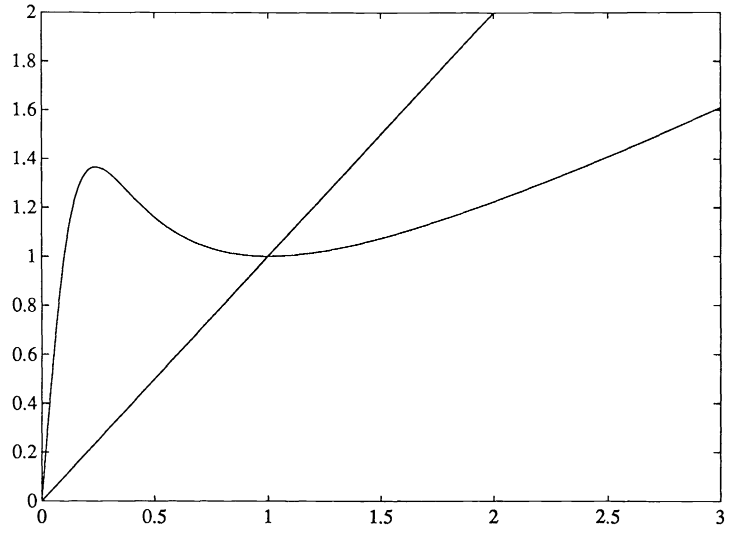


FIG. 3. $f=2.1$.

With this class of iteration functions we get for the small σ_i' corresponding to $\sigma_i < 1$ the bounds:

$$F(\sigma_n) \leqq \sigma_i' \leqq F(x_e)$$

or approximately

$$F'(0)\sigma_n \sim \frac{\sigma_n}{\varepsilon} \leqq \sigma_i' \leqq \frac{1}{2}\frac{1}{\sqrt{2\varepsilon}}.$$

Again here for any $\alpha > 0$ we have global convergence.

(4) $f = 2$ (see Fig. 4). Higham's nonaccelerated iteration. Large values $\sigma_i > 1$ are decreased by a factor 2. Small $\sigma_i < 1$ are heavily amplified:

$$\sigma_i' \sim \frac{1}{2\sigma_i}$$

so that after the first iteration all the singular values are greater than 1 and in the following iteration they are decreased slowly by a factor of 2. For any $\alpha > 0$ we have global convergence.

(5) $1 < f < 2$ (see Fig. 5, e.g., for $f = 1.4$). F has a pole at

$$x_p = \sqrt{\frac{2-f}{f}}$$

and $0 < x_p < 1$. The iteration converges for $x_p < \sigma_i$. For starting values $\sigma_i > 1$ the iteration converges monotonically and slowly to the desired limit. Therefore by scaling with $\alpha = 1/\sigma_n$ we have global convergence. For $1 < f < \frac{5}{3}$ the fixed-point zero is attractive, i.e., values $0 \leqq \sigma_i < x_p$ converge to 0. This could be used in special applications where the matrix A does not have full rank and where one would like the zero singular values to remain zero and not converge to 1.

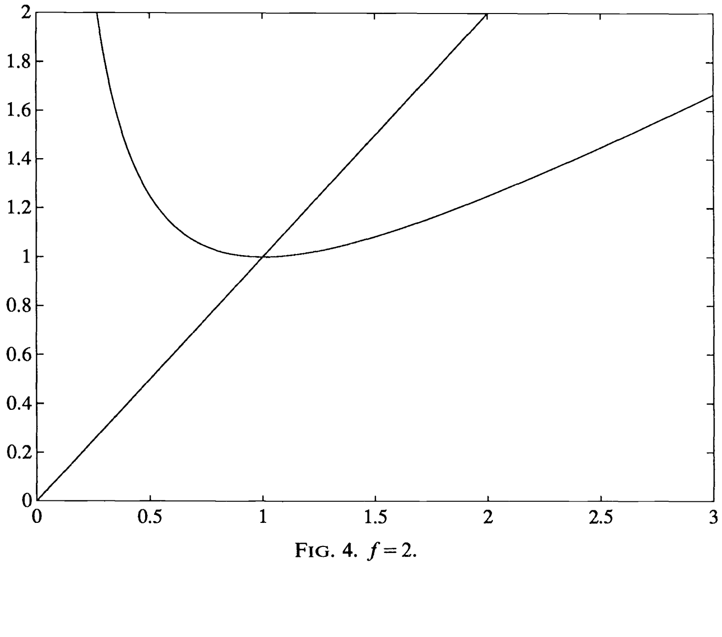


FIG. 4. $f = 2$.

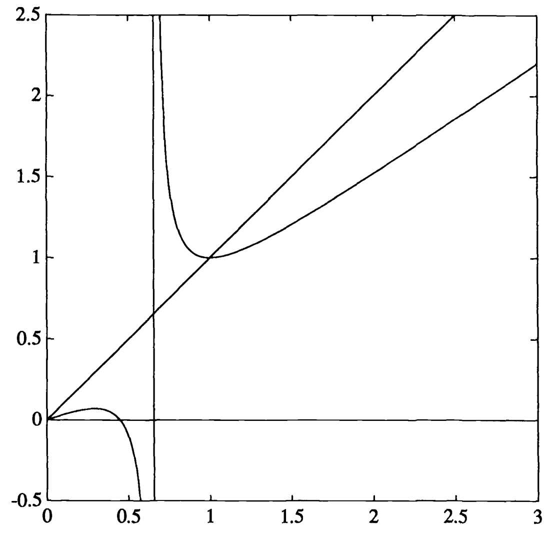


FIG. 5. $f=1.4$.

(6) $0 \leqq f < 1$ (see Fig. 6, e.g., for $f=0.8$). Those methods converge only for $0 \leqq \sigma_i \leqq x_n = \sqrt{3-2f}$. Therefore one has to scale with $\alpha < \sqrt{3-2f}/\sigma_1$ to get global convergence. The pole is at $x_p > 1$. For $f=0$ we receive Björck and Bowie's iteration (for $p=1$), and F is a polynomial of degree 3 in this case.

(7) $f<0$ (see Fig. 7, e.g., for $f=-2$). The situation is quite similar to the last case. The only difference is that no pole exists anymore. We have convergence for $0 \leqq \sigma_i \leqq x_n = \sqrt{3-2f}$ and global convergence by scaling with $\alpha < \sqrt{3-2f}/\sigma_1$.

5. An algorithm for general rank defective matrices. By choosing an iteration method (4.21) it is important that the condition numbers of the matrices $C_k = (f-2)I + fX_k^TX_k$ be as small as possible. If σ_n and σ_1 denote the smallest, respectively,

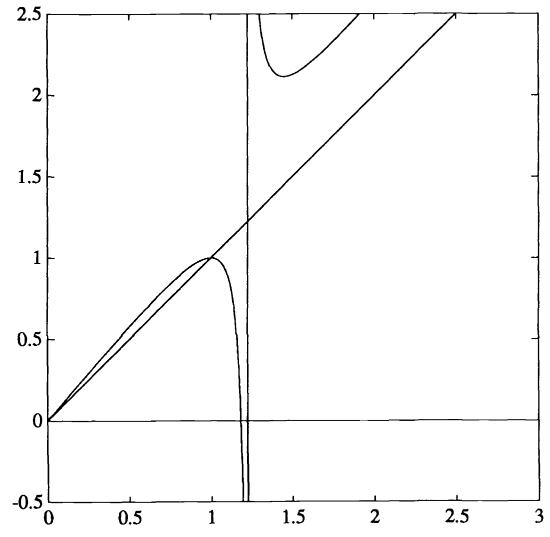


FIG. 6. $f=0.8$.

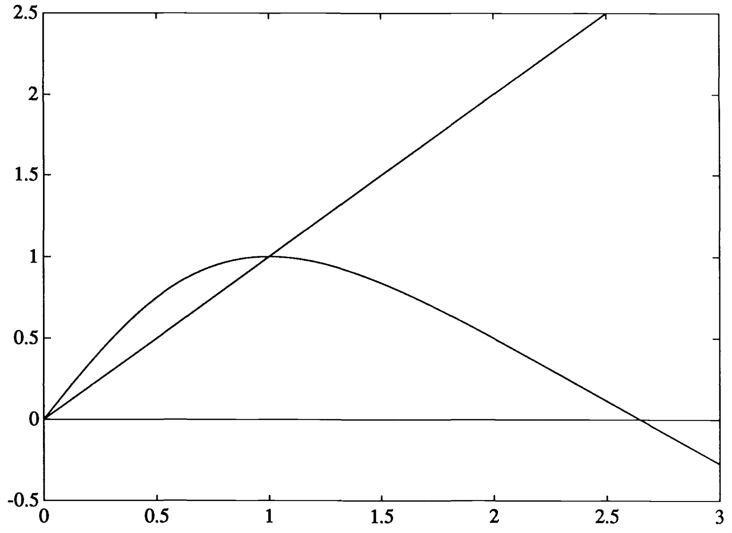


FIG. 7. $f=-2$.

the largest singular value of X_k, then C_k has the condition number

$$\kappa(C_k)=\left|\frac{f-2+f\sigma_1^2}{f-2+f\sigma_n^2}\right|. \tag{5.24}$$

Let us assume that we have scaled the matrix $X_0=A/\|A\|$ so that $\sigma_1\approx 1$ and $\sigma_n\approx 0$ (rank defective). The condition of C_0 is then

$$\kappa(C_0)=\left|\frac{2f-2}{f-2}\right|. \tag{5.25}$$

In this case C_0 gets ill-conditioned if $f=2+\varepsilon$ is close to 2. However, in order to make the tiny singular values grow fast, one would like to use such values for f!

Let ε denote the machine precision. Solving the linear system in the first iteration will introduce errors proportional to $\kappa(C_0)\times\varepsilon$. If we want the result accurate to the convergence tolerance $\delta>\varepsilon$, then we have to choose

$$f=2+\varepsilon/\delta,$$

and we have to perform about

$$steps=\log(\varepsilon)/\log\left(\frac{\varepsilon}{\delta}\right)$$

iteration steps to increase the smallest singular values approximately equal to ε to the size of 1. We do not allow ε/δ to be smaller than 10^{-5}, so that the singular values cannot exceed 111.8 by (4.23). A reducing step with $f=F(x_e)$ is then followed by some final steps with Halley's cubically convergent method. This strategy is used in the algorithm *pd* listed in the appendix.

6. Examples. ε denotes the machine precision ($\varepsilon=2.2204e-16$).

(1) Random 20×10 matrix with rank 2. The nonzero singular values were $\sigma_1=7.3883$, $\sigma_2=1.1665$. The call $[U, H]=pd(A,\delta,0)$ gave the results of Table 2:

TABLE 2

δ	10ε	100ε	$1{,}000\varepsilon$	$10{,}000\varepsilon$
f (#it)	2.1 (17)	2.01 (9)	2.001 (6)	2.0001 (5)
total #it	20	13	11	10
$\|UH-A\|_2/\|A\|_2$	$3.4811e-15$	$2.9104e-14$	$2.9616e-13$	$2.4567e-12$
$\|U^TU-I\|_2$	$4.7161e-16$	$4.5502e-16$	$4.7110e-16$	$4.4369e-16$

In all cases two eigenvalues of the computed H were equal to the two singular values of A and the others were on rounding error level $|\lambda(A)|<2.3e-15$.

(2) $A=gallery(5)$ MATLAB [8]. This matrix has rank 4 and the singular values: $1.0104e+05$, 1.6795, 1.4628, 1.0802, 0. In the following table we display the singular values of X_k for the call $pd(A, 100*\varepsilon, 0)$. The first nine steps are executed with $f=2.01$. Then one step is done with $f=3.535$ and in the last three steps $f=3$. Table 3 shows how the smallest singular value grows with a factor of about 100 per step.

TABLE 3

σ_1	σ_2	σ_3	σ_4	σ_5
1.1449	$2.8959e-03$	$2.5224e-03$	$1.8625e-03$	$6.4920e-15$
1.0090	$2.9488e-01$	$2.5695e-01$	$1.8985e-01$	$6.6246e-13$
2.4318	1.9554	1.7665	1.0000	$6.7574e-11$
1.4173	1.2308	1.1643	1.0000	$6.8925e-09$
1.0606	1.0213	1.0114	1.0000	$7.0304e-07$
1.0017	1.0002	1.0001	1.0000	$7.1710e-05$
1.0000	1.0000	1.0000	1.0000	$7.3144e-03$
1.0000	1.0000	1.0000	1.0000	$7.3817e-01$
1.0452	1.0000	1.0000	1.0000	1.0000
1.0000	1.0000	1.0000	1.0000	$9.9981e-01$
1.0000	1.0000	1.0000	1.0000	1.0000
1.0000	1.0000	1.0000	1.0000	1.0000
1.0000	1.0000	1.0000	1.0000	1.0000

The accuracy is given as predicted by the choice of f by $\|UH-A\|_2/\|A\|_2=1.4291e-14$ and $\|U^TU-I\|=3.3514e-16$. The smallest eigenvalue of H was $6.0596e-13$.

(3) $A=$ columns 1: 20 of the 50×50 Hilbert matrix. (See Table 4.) The singular values, computed by MATLAB, are $\sigma_1=1.9774$, $\sigma_{20}=6.4300e-18$.

TABLE 4

	#*it*	$\|UH-A\|_2/\|A\|_2$	$\|U^TU-I\|_2$
$pd(A, 100*\varepsilon, 0)$	13	$3.6110e-14$	$4.5965e-16$
generalized Higham	10	$1.3529e-05$	$1.6241e-16$

pd used $f=2.01$ for the first nine steps. The computed H had two negative eigenvalues on rounding error level ($-7.3301e-18$ and $-1.4956e-17$). For generalized Higham the error warning "matrix singular" occurred twice due to poor condition of the matrix; therefore the results are inaccurate.

(4) $A=$ Hilbert (5×5). Since A is symmetric and positive definite, we should have $U=I$ and $H=A$. The results are given in Table 5.

TABLE 5

	#it	$\|H-A\|_2$	$\|U-I\|_2$
Higham	8	7.8163e−17	2.9403e−15
generalized Higham	8	1.3569e−11	9.5408e−09
$pd(A, \varepsilon, rcond(a))$	11	8.8959e−16	1.6620e−14

The loss of accuracy with generalized Higham is again due to squaring the condition number by this method. *rcond* is LINPACK estimate of the reciprocal condition number.

(5) $A=$ Hilbert (30×30). (See Table 6.) The iteration accuracy was $\delta = 10\varepsilon$ in all cases.

TABLE 6

	#it	$\|H-A\|_2$	$\|U-I\|_2$	$\|U^TU-I\|_2$
Higham	10	9.8234e−13	1.9994	6.4666e−16
generalized Higham	10	8.1918e−04	1.9893	2.2865e−16
$pd(A, \delta, rcond(a))$	21	3.6630e−15	2	7.0639e−16

Due to the numerical singularity of the matrix we get one warning from MATLAB for Higham's and two warnings for the generalized Higham's method. Though $U \neq I$ the results of *pd* are good: U is orthogonal to machine precision, and A equals H as well as one can expect (eight eigenvalues of H were negative, however, in modulus smaller than $5.9e-17$). Numerically this matrix behaves as if it were rank defective. Therefore U is not unique [5] and can be different from I.

(6) We have tested *pd* successfully for various matrices. We could find the following example where it failed. Let $G = gallery(5)$ MATLAB [8]. We construct the matrix A as

$$A = \begin{pmatrix} G & G \\ G & G \end{pmatrix}.$$

By calling $pd(A, 10 * \varepsilon, 0)$ we observe that in 20 iterations only five singular values grow to 1, the other five remain on the rounding error level. It seems that for this special case the rounding errors are too correlated. If we form UU^T we get

$$UU^T = 0.5\begin{pmatrix} I & I \\ I & I \end{pmatrix}.$$

Appendix. Listing of the program *pd*.

```
function [U, H] = pd(A, delta, rcon)
% pd    Iterative computation of the polar
%       decomposition of an arbitrary matrix
%
%       Input arguments:
%        A    = m-by-n matrix
%         delta = convergence tolerance, <1
%          rcon = estimate for the inverse condition
%                 number: sigma_n/sigma_1 (zero
%                 for rank deficient matrices)
%        [U, H] = pd(A, delta, rcon)
%
[m, n] = size(A);       % Size of the matrix A
```

```
k=0;                   % Counter for iteration
% Data adjustment for delta
if delta<eps, delta=eps; end,
if delta>0.01, delta=0.01; end,
e=max([rcon,eps])
if delta<=10*eps,
    f=2.1, ma=3,
    steps=round(log(e)/log(0.1))+1,
elseif delta>100000*eps,
    f=2.00001, ma=100,
    steps=round(log(e)/log(0.00001))+1,
else f=2+eps/delta, ma=1/sqrt(8*eps/delta),
    steps=round(log(e)/log(eps/delta))+1,
end
if m>=n
    X_k=0;
    X_k1=A/norm(A,inf)*sqrt(n);
    while norm(X_k1-X_k,1)>delta*norm(X_k1,1)
        X_k=X_k1; k=k+1;
        AA=X_k'*X_k;
        X_k1=X_k*((2*f-3)*eye(n)+AA)..
                    /((f-2)*eye(n)+f*AA);
        if k==steps,
            f=ma,
        elseif k>steps,
            f=3,
        end,
    end
else % m<n
    X_k=0;
    X_k1=A/norm(A,inf)*sqrt(n);
    while norm(X_k1-X_k,1)>delta*norm(X_k1,1)
        X_k  =X_k1; k=k+1;
        AA   =X_k*X_k';
        X_k1=((2*f-3)*eye(m)+AA)..
               /((f-2)*eye(m)+f*AA)*X_k;
        if k==steps,
            f=ma,
        elseif k>steps,
            f=3,
        end,
    end
end
U=X_k1;
H=(U'*A+A'*U)/2;
```

REFERENCES

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

[2] A. BJÖRCK AND C. BOWIE, *An iterative algorithm for computing the best estimate of an orthogonal matrix*, SIAM J. Numer. Anal., 8 (1971), pp. 358-364.

[3] J. DONGARRA, J. DU CROZ, I. DUFF, AND S. HAMMARLING, *A set of level* 3 *basic linear subprograms*, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, August 1988.

[4] N. J. HIGHAM, *Computing the polar decomposition—With applications*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1160-1174.

[5] N. J. HIGHAM AND R. S. SCHREIBER, *Fast polar decomposition of an arbitrary matrix*, Tech. Report, Department of Computer Science, Cornell University, Ithaca, NY, 1988.

[6] W. GANDER, *On Halley's iteration method*, Amer. Math. Monthly, 92 (1985) pp. 131-134.

[7] G. GOLUB AND C. VAN LOAN, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, 1983.

[8] C. MOLER, J. LITTLE, AND S. BANGERT, PRO-MATLAB *User's Guide*, The Math. Works Inc., Sherborn, MA, 1987.

[9] R. SCHREIBER AND B. PARLETT, *Block reflectors: Theory and computation*, SIAM J. Numer. Anal., 25 (1988), pp. 189-205.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 6, pp. 1116-1135, November 1990

SOLVING SYSTEMS OF NONLINEAR EQUATIONS ON A MESSAGE-PASSING MULTIPROCESSOR*

THOMAS F. COLEMAN† AND GUANGYE LI‡

Abstract. Parallel algorithms for the solution of dense systems of nonlinear equations on a message-passing multiprocessor computer are developed. Specifically, a distributed finite-difference Newton method, a multiple secant method, and a rank-1 secant method are proposed. Experimental results, obtained on an Intel hypercube, indicate that these methods exhibit good parallelism.

Key words. systems of nonlinear equations, hypercube computer, message-passing multiprocessor, secant method, finite-difference Newton method, parallel algorithms

AMS(MOS) subject classifications. 65H10, 65K05, 65K10

1. Introduction.

In this paper we investigate parallel algorithms, tailored to the hypercube multiprocessor context, for the solution of systems of nonlinear equations

$$\text{solve } F(x)=0 \tag{1}$$

where $F: R^n \to R^n$. Component i of F is denoted by f_i. We assume that F is differentiable; let $J(x)$ denote the Jacobian matrix evaluated at point x.

Our implementations are specific to a hypercube multiprocessor; however, the algorithmic ideas are applicable more generally. In particular, the parallel algorithms presented here can be tailored to any multiprocessor computer provided that the communication topology allows for efficient "fan-in" and "fan-out" operations and the processors themselves have significant local memory. Furthermore, some of our proposed algorithms—multisecant update, triangular solve—are most meaningful when a "ring" communication pattern is used; hence, the topology of the multiprocessor should allow for a ring embedding. Finally, we remark that we always assume that the dimension of the problem n is greater than the number of processors p; indeed, the algorithms uniformly become more efficient as n/p increases.

Our ultimate interest is in large sparse problems; however, in this paper we restrict our attention to the case in which the Jacobian matrix is assumed to be dense.

In a nutshell, this paper represents our attempt to parallelize the popular globalized Newton-like approaches to (1), such as secant and finite-difference Newton methods with a dogleg step or linesearch procedure. Consequently, the algorithms under consideration actually solve the structured minimization problem

$$\text{minimize}\,\{f(x): f(x)=\tfrac{1}{2}F(x)^T F(x)\}. \tag{2}$$

Obviously a solution to (1) is also a solution to (2); unfortunately, the converse is not always true. Nevertheless, such algorithms often are used successfully to obtain solutions to (1).

* Received by the editors January 1, 1988; accepted for publication (in revised form) October 17, 1989.

† Computer Science Department and Center for Applied Mathematics, Cornell University, Ithaca, New York, 14853. This research was partially supported by the Applied Mathematical Sciences Research Program (KC-04-02) of the Office of Energy Research of the U.S. Department of Energy under grant DE-FG02-86ER25013.A000.

‡ Computer Center, Jilin University, People's Republic of China. This research was partially supported by the U.S. Army Research Office through the Mathematical Sciences Institute, Cornell University, Ithaca, New York 14853.

We divide the world into two classes of functions: functions F that are most conveniently evaluated as a single entity (i.e., a single subroutine evaluates the entire vector function F, sequentially), and functions F that can be evaluated in a distributed, parallel fashion. In this paper, to be concrete, we restrict the latter category to functions that conveniently separate into component sequential subroutines for each f_i, $i = 1:n$. We call such functions *row-separable.*[1]

If F is to be evaluated as a single entity, then our approach is to distribute copies of the F-evaluation subroutine to all the processors. We then assume that any node (processor) can evaluate $F(x)$, given x, with no other communication necessary. Note that the usual (rank-1) secant method cannot be parallelized in any obvious way since the evaluation of $F(x)$ is not distributed. If the evaluation of F is cheap relative to the other computations (e.g., matrix updating, triangular solves, ...) then this poses no problem—let one node evaluate F while the others remain idle; however, if F is relatively expensive, then it is not clear how to effectively parallelize the rank-1 secant method. For this reason we have developed the *multisecant method* in which each processor evaluates F at a different point (or perhaps several different points). The result is a rank-q update, where q is a multiple of the number of processors available. This approach is discussed in § 5. In the extreme case, when each node is evaluating F at many different points, the multiple secant method resembles a parallel finite-difference Newton method. The latter approach is explored in § 4.

When F is row-separable the evaluation of $F(x)$ can be done in parallel. This allows for an efficient parallel version of the (globalized) rank-1 secant method. The crucial problem here is the design of the effective parallel QR-updating scheme. We discuss this in § 3.

Next we briefly describe the salient features of a hypercube computer. See Wiley [15], for example, for more information.

A message-passing multiprocessor consists of several independent processors connected by communication links. Each processor has significant local memory. (For example, the Intel iPSC with extra memory boards has approximately four megabytes of available memory, per node.) There is also a host computer, connected to one or several of the nodes (processors), whose purpose is to load programs and data onto the nodes of the cube, as well as collect the answer; we take the view that the host does not participate in intermediate computations.

Each processor supports two message-passing primitives: *send* and *receive.* When a node *sends* an array, it is transported through a sequence of nodes and communication links—the sequence is usually determined by the operating system—until the target node is reached. Upon executing a *receive*, a node checks to see if a new message is in its buffer. If so, the message is read and execution continues; if not, execution is suspended (on the receiving node only) until a message arrives.

A hypercube computer is a particular kind of message-passing multiprocessor. Specifically, the name refers to the topology defined by the communication links. A zero-dimensional hypercube, or 0-cube, is a single processor. To construct a 1-cube (i.e., 2 nodes), join two 0-cubes with a single communication link. In general, construct an m-cube (i.e., 2^m nodes) from two $m-1$ cubes: find a one-to-one correspondence between the nodes in each cube and add a communication link between each pair.

[1] We restrict our attention to row-separable functions to provide specific explicit algorithms, and for purposes of implementation and experimentation. However, the ideas and algorithms developed for row-separable functions are easily adapted to the general situation: i.e., functions that can be evaluated in a distributed, parallel manner.

The hypercube topology allows for a number of interesting properties (e.g., [14]). A particularly important one, for our purposes, is that a spanning tree of depth m can be embedded in an m-cube, *rooted at any node.* This allows for the efficient implementation of a number of global operations. For example, a node can send information to every other node in m "timesteps." This is usually called a *broadcast* or *fan-out* operation. Alternatively, a vector distributed over the cube can be collected onto any single node (the target node) in m "timesteps" using a spanning tree rooted at the target node. This is often called a *fan-in* operation.

Each node has a unique name *myid*, which is a number in the range $[0, p-1]$ where p is the number of processors; each node is aware of its own name. We use two labelings, or assignment of node names. The first is the *natural* ordering, which is an assignment of integers in $[0, p-1]$ such that each neighbor of node i (a neighbor of node i is a node connected to i by a single communication link) differs by a single bit in its binary representation of its name. For example, the neighbors of node 5, in a 4-cube, are nodes 4, 7, 1, and 13. A *ring* ordering is also used. In this case node i is connected by single communication links to node $(i-1) \bmod p$ and node $(i+1) \bmod p$. An m-cube always allows for an embedding of a ring on 2^m nodes.

Experimental results reported in this paper were obtained using the Cornell Theory Center 16-node Intel iPSC hypercube under Xenix 286 release 3.4 of the host operating system and iPSC release 3.0 of the node operating system. The nodes were equipped with extra memory boards yielding approximately four megabytes of available memory per node. All our programs were written in Fortran.

2. The sequential secant algorithm. We begin by summarizing a simplified version of the Minpack [11] sequential secant algorithm. This algorithm, in turn, is based on the work of Powell [12].

Suppose x_c is the current approximation to a solution of (2) and define $F_c \overset{\text{def}}{=} F(x_c)$. Let B_c be the current Jacobian approximation and let $B_c = Q_c R_c$ be the QR-factorization of B_c. Assume B_c to be nonsingular.

A *trial step* s_c is computed by approximately solving the trust region problem

$$\text{minimize}\,\{\|F_c + B_c s\|_2^2 \colon \|s\|_2 \leqq \Delta_c\} \tag{3}$$

where Δ_c is the current radius of the trust region. The approximate solution s_c is obtained by further restricting (3): specifically, s_c solves the problem

$$\text{minimize}\,\{\|F_c B_c s\|_2^2 \colon \|s\|_2 \leqq \Delta_c,\ s \in P_c\} \tag{4}$$

where P_c is a piecewise linear path defined as follows: First, connect x_c to the Cauchy point, $x_c + s_c^{\text{Cauchy}}$, where

$$s_c^{\text{Cauchy}} \overset{\text{def}}{=} -\frac{\|B_c^T F_c\|_2^2}{\|B_c B_c^T F_c\|_2^2} B_c^T F_c, \tag{5}$$

and then connect $x_c + s_c^{\text{Cauchy}}$ to the Newton point, $x_c + s_c^{\text{Newton}}$, where

$$s_c^{\text{Newton}} \overset{\text{def}}{=} -B_c^{-1} F_c. \tag{6}$$

The computation determining the trial step s_c boils down to the algorithm in Fig. 1. (Note. *newx* is a logical input parameter. If *newx* = *false* then all quantities in step 1 have not changed since the previous call; otherwise, *newx* = *true* and all quantities have changed values.)

The possible correction s_c determined by algorithm *Dogleg* is accepted (i.e., $x_+ \leftarrow x_c + s_c$) only if $\|F(x_c + s_c)\|_2 < \|F(x_c)\|_2$. If s_c is not accepted then Δ_c is reduced and step 2 of algorithm *Dogleg* is repeated.

```
{1: Compute Cauchy and Newton Steps}
If {newx} then
    {Compute $s_c^{\text{Cauchy}}$ (given $Q_c$, $R_c$)}
    $u \leftarrow Q_c^T F_c$;
    $g \leftarrow R_c^T u$;
    $w \leftarrow Q_c R_c g$;
    $s_c^{\text{Cauchy}} \leftarrow -\frac{\|g\|_2^2}{\|w\|_2^2} g$;
    {Compute $s_c^{\text{Newton}}$}
    Solve $R_c s_c^{\text{Newton}} = -u$;
Endif

{2: Solve (4)}
If $\{\|s_c^{\text{Newton}}\|_2^2 \leqq \Delta_c\}$ then $s_c \leftarrow s_c^{\text{Newton}}$

Elseif $\{\|s_c^{\text{Cauchy}}\| \geqq \Delta_c\}$ then $s_c \leftarrow \left(\frac{\Delta_c}{\|s_c^{\text{Cauchy}}\|_2}\right) s_c^{\text{Cauchy}}$

Else $s_c \leftarrow s_c^{\text{Cauchy}} + \alpha(s_c^{\text{Newton}} - s_c^{\text{Cauchy}})$
    where $\alpha$ is the positive root of the quadratic equation
    $\|s_c^{\text{Cauchy}} + \alpha(s_c^{\text{Newton}} - s_c^{\text{Cauchy}})\|_2^2 = \Delta_c^2$
Endif
```

FIG. 1. *Algorithm Dogleg.*

There must also be a mechanism for increasing Δ_c so that progress is not impeded by unnecessarily small steps. This is accomplished by comparing the predicted reduction to the actual reduction. If this ratio, *ratio*, is sufficiently large then Δ_c is increased.

Besides updating x and Δ, it is necessary to evaluate $F(x_c + s_c)$ and update the QR-factorization of B to reflect the rank-1 secant update (due to Broyden [2]). We will not go into the QR-updating details here; however, orthogonal rotations can be used to stably perform this update using a total of $26n^2$ floating point operations (e.g., [5]). The algorithm in Fig. 2 is a formalization of the ideas expressed above.

The algorithm described in Fig. 2 represents a simplified version of the Minpack implementation. For example, Minpack will refresh B by finite-differences when it appears that convergence is not proceeding rapidly enough. In addition, Minpack will modify R if singularity is detected. Furthermore, the Minpack "ratio test" and subsequent adjustment of Δ is somewhat more complicated. We will not spell out these details in this description since they do not bear significantly on questions of parallelization. Subscript "c" denotes current: e.g., x_c refers to the current point. Subscript "+" denotes the updated (new) quantity: e.g., x_+ is the new value of x, $x_+ \leftarrow x_c + s_c$.

3. Parallel secant method for row-separable functions.

3.1. The algorithm. As mentioned in § 1, a row-separable function F is defined to be one in which it is convenient to have available a separate subroutine to evaluate each $f_i(x)$, $i = 1:n$. Assuming row-separability (see footnote 1), we now develop a parallel *Secant/Dogleg* secant method for (1) based on the sequential secant method described in § 2.

Our general approach is to distribute data and functions around the cube so that the work in the computationally intensive steps in algorithm *Secant/Dogleg* is well distributed; we are averse to redistributing information if it can be avoided. We make no attempt to parallelize steps that involve relatively insignificant computational work.

```
{0: Initialize}
Choose $x_c$, evaluate $F(x_c)$, determine $B(x_c)$ by finite differences;
Compute $B(x_c) = Q_cR_c$
newx ← true;

{1: Attempt to find a zero of $F(x)$}
Repeat
    Determine $s_c$ by Algorithm Dogleg(newx);
    Evaluate $F(x_c + s_c)$;

    {Compute ratio}
    $actred \leftarrow 1 - \frac{\|F(x_c + s_c)\|_2^2}{\|F_c\|_2^2}$;
    $pred \leftarrow 1 - \frac{\|F_c + Q_cR_cs_c\|_2^2}{\|F_c\|_2^2}$;
    $ratio \leftarrow \frac{actred}{pred}$;

    {Update x}
    If {$ratio \leqq .0001$} then $x_+ \leftarrow x_c$
    Else $x_+ \leftarrow x_c + s_c$ Endif

    {Update Δ}
    If {$ratio \leqq \frac{1}{4}$} then $\Delta_c \leftarrow \frac{1}{2}\Delta_c$
    Elseif {$ratio \leqq \frac{3}{4}$} then $\Delta_+ \leftarrow \Delta_c$
    Else $\Delta_+ \leftarrow 2\Delta_c$ Endif

    {Update B, newx}
    If {$x_+ \neq x_c$} then
        newx ← true;
        Update $Q_cR_c \rightarrow Q_+R_+$ to reflect the rank-1 change:
            $B_+ \leftarrow B_c + \frac{([F_+ - F_c] - B_cs_c)s_c^T}{s_c^Ts_c}$;
    Else newx ← false
    Endif
Until {convergence}
```

FIG. 2. *Algorithm Secant.*

From our perspective the significant steps in algorithm *Dogleg* are the matrix-vector multiplies and the upper triangular solve in Step 1; Step 2 is relatively insignificant and can be performed on a single node. Beyond the call to *Dogleg*, the significant steps in algorithm *Hybrid* are: the initial finite-difference approximation B, the initial QR-factorization, the evaluation of $F(x_c + s_c)$, matrix-vector multiplies in the computation of *prered*, and the update of the QR-factorization.

We distribute F as follows: For $i = 0: p-1$, node i is assigned component subroutine f_j for each $j = i + 1 (\text{mod}\, p)$, $1 \leqq j \leqq n$. Therefore, to evaluate $F(y)$ each node must just evaluate its resident component functions: the simple node program is illustrated in Fig. 3.

```
$k \leftarrow myid + 1$;
**While** $\{k \leqq n\}$ **do**
    Evaluate $f^k(y)$
    $k \leftarrow k + p$
**Endo**
```

FIG. 3. *Algorithm* F-*evaluate* (*node program*).

The initial finite-difference determination of B can be accomplished in parallel using algorithm F-*Evaluate* as a subroutine. However, it turns out to be convenient to have the initial B distributed by columns; therefore, given our distribution of F, some communication is required within the parallel finite-difference method. The basic idea is to use algorithm F-*Evaluate* to parallelize the usual column-oriented finite-difference scheme. This is based, in turn, on the approximation to the jth column of the Jacobian matrix,

$$J(x)e_j \cong \frac{F(x+\tau e_j)-F(x)}{\tau} \tag{7}$$

where e_j is the jth column of the identity matrix and τ is an appropriate positive scalar. The node program for the parallel finite-difference approximation is given in Fig. 4. It is assumed that every node has a copy of x, the current point, and τ, the differencing scalar.

Note that each node determines some of the components of column j; the "Fan-in" step collects column j onto node $(j-1) \bmod p$ where it is stored.

The initial QR-factorization of B can be accomplished by a parallel column-oriented algorithm based on orthogonal Householder transformations. Moler [9] has described the framework for such an algorithm in which the column-distributed matrix B is overwritten with R and the Householder vectors that define Q. However, our rank-1 updates require an explicit representation of Q; therefore, we have modified Moler's Algorithm to produce a *row-distributed* Q-matrix, while overwriting B with the *column-distributed* matrix R. This modification is rather straightforward and we will not describe it here; however, in Table 1 of § 3.2 we do provide results of numerical experiments designed to measure parallel efficiency.

There are numerous matrix-vector multiplies in algorithm *Secant/Dogleg* involving matrices Q, R, Q^T, and R^T: we have built our (straightforward) routines based on the communication utility routines provided by Intel [10]. Design of an efficient parallel triangular solver turns out to be a difficult problem. Nevertheless, there has been significant recent progress (e.g., [6]-[8], [13]); we use the algorithm of Li and Coleman [8] in our implementation.

It remains to consider the QR-updating step. Indeed, our decision to crossthread Q and R—i.e., distribute Q by rows and R by columns—was arrived at with this step in mind. Hence we assume that if $j-1=i \bmod p$, $1 \leqq j \leqq n$, then node i houses row j

```
**For** $j = 1$: $n$ **do**
    {Estimate column $j$}
    $y \leftarrow x + \tau e_j$;
    *F-Evaluate* $(y) \rightarrow w$;
    Participate in Fan-in of $w \rightarrow$ node $(j-1) \bmod p$;
**Endo**
```

FIG. 4. *Algorithm* J-*evaluate* (*node program*).

of Q and column j of R. Let

$$B_+ = QR + rs^T = Q(R + \bar{r}s^T) \tag{8}$$

where $\bar{r} = Q^T r$. Assume that $\bar{r}$ is stored on a single node, node 0, say; let s^T be distributed, by column, to conform to the distribution of R.

Recall that the usual sequential algorithm [4] introduces zeros in $\bar{r}$ by applying orthogonal rotations to its rows, from bottom to top. Each rotation is applied, in turn, to the two corresponding rows of R and columns of Q. But the distribution we have chosen for Q and R is ideal for parallelization: each node contains a segment of the rows of R (and columns of Q) being rotated. Hence, the work involved in the rotation is well distributed.

Upon completion of the step described above we have $B_+ = \hat{Q}\hat{R}$ where $\hat{Q}$ is orthogonal and $\hat{R}$ is upper-Hessenberg. Next we must reduce $\hat{R}$ to upper triangular form. This is done using orthogonal rotations applied from top to bottom. Rotation G_i is applied to rows i, $i+1$ of $\hat{R}$ as well as columns i, $i+1$ of Q (for $i = 1\colon n-1$). Rotation G_i involves computations that can be done concurrently because each node has a segment of rows (columns) i, $i+1$ of $\hat{R}(\hat{Q})$.

Figure 5 provides the detailed algorithm. For each node k, let

$$I(k) = \{1 \leqq i \leqq n\colon i - 1 = k \bmod p\}. \tag{9}$$

Remark. As we have demonstrated, the parallelization of the *Secant/Dogleg* Algorithm is fairly straightforward under a row-separability assumption. The two most

```
{Reduce to Upper Hessenberg form}
For $i = n-1\colon 1(-1)$ do
    If {$myid = 0$} then
        Determine $G_i$; {Givens rotation defined by $\bar{r}_{i+1}$, $\bar{r}_i$}
        Apply $G_i$ to $\bar{r}$;
        Broadcast $G_i$;
    Endif

    For each $k \in I(myid)$ do
        Rotate elements in rows $i$, $i+1$ of column $k$ of $R$, using $G_i$;
        Rotate elements in columns $i$, $i+1$ of row $k$ of $Q$, using $G_i$;
    Endo
Endo
If {$myid = 0$} then broadcast $\alpha \overset{\text{def}}{=} \bar{r}_1$ Endif

Add $\alpha \times s$ to first row of $R$;

{Reduce $R$ back to upper triangular form}
For $i = 1\colon n-1$ do
    If {$myid = (i-1) \bmod p$} then
        Determine $G_i$, based on $R_{i+1,i}$, $R_{i,i}$; {Givens rotation}
        Broadcast $G_i$;
    Endif

    For each $k \in I(myid)$ do
        Rotate elements in rows $i$, $i+1$ of column $k$ of $R$, using $G_i$;
        Rotate elements in columns $i$, $i+1$ of row $k$ of $Q$, using $G_i$;
    Endo
Endo
```

FIG. 5. *Algorithm* QR-*update* (*node program*).

challenging steps are the triangular solve and QR-update. Indeed, it is possible to avoid these two steps altogether if we recur B^{-1} instead of B. This is quite possible (e.g., [3]) though from a numerical point of view it is probably preferable to update B. There are two major reasons we do not pursue this possibility here. First, one of our goals is to determine if the Minpack Algorithm, which includes updating QR-factors, can be efficiently parallelized. Second, we maintain an eye toward the sparse case in this development: in general, J^{-1} is dense when J is sparse and therefore recurring an approximation to J^{-1} is unreasonable in the large sparse situation.

3.2. Numerical experiments. In Table 1 we present timing results reflecting the performance of the QR-updating Algorithm described in Fig. 5, and, for comparison, the QR-factorization routine (which we have not described in detail here) and the triangular solve. Note that even though both the QR-update and the triangular solve are $O(n^2)$ operations, the efficiency of the rank-1 update is much better than the triangular solve efficiency. This is due to the constant factors involved: i.e., the QR-update requires $26n^2$ arithmetic operations, whereas the triangular solve involves $1n^2$ arithmetic operations. Moreover, the efficiency of the update is not dramatically worse than for the full factorization that, in turn, exhibits close to optimal megaflop rate. The near-maximum efficiency of the QR-factorization is due to the intensive computational work required (recall that the orthogonal matrix Q is being explicitly formed).

In Table 2 we present the running times for our parallel implementation of the secant algorithm described in § 2. Our stopping criterion was $\|F\| \leqq 10^{-8}$; the "update Δ" step was modified to conform with the Minpack code. Problem 15 is the well-known extended Rosenbrock function; the other problems were chosen from the Minpack collection of nonlinear equation problems.

To provide a measure of speedup, in Table 3 we have divided the numbers in Table 2 by the sequential running times of a *modified* Minpack code running on a

TABLE 1
Timing results for the parallel secant update, $p = 16$.

n	QR time	QR mflps	Update time	Update mflps	Solve time	Solve mflps
100	7.2	.458	.995	.261	.70	.028
200	51.3	.52	2.6	.39	1.3	.06
300	166.0	.54	5.6	.46	2.2	.08

TABLE 2
Running times for the parallel secant algorithm.

Problem	p	$n = 50$	$n = 100$	$n = 300$
9	1	20.5	128.27	2494.7
9	16	5.54	15.6	162.0
10	1	60.14	418.6	10264.3
10	16	9.4	36.7	643.9
14	1	92.6	397.0	5210.3
14	16	40.19	89.2	468.7
15	1	193.14	945.1	13121.2
15	16	85.7	227.9	1445.5

TABLE 3
Speedup for the parallel secant method, $p = 16$.

Problem	$n = 50$	$n = 100$	$n = 300$
9	3.8	8.2	15.4
10	6.4	11.4	15.9
14	2.3	4.5	11.1
15	2.25	4.1	9.1

single processor. The Minpack code was modified to force secant updates (after the initial finite-difference Jacobian estimation). Moreover, the Minpack stopping criteria were replaced by the rule mentioned above. Hence the two codes produce exactly the same sequence of x-iterates (we verified that this claim held on the four test problems in question).

Remarks. (1) Obviously speedups improve as n increases. This is due to the increase in distributed work to be performed.

(2) Problems 9 and 10 require only unit steps each iteration; on the other hand, problems 14 and 15 require many nonunit steps—many trial steps are rejected. It is this fact, *in combination* with the fact that function evaluations are *extremely cheap* that accounts for the relatively poor parallel efficiency demonstrated on problems 14 and 15.

To support this claim numerically, we have artificially increased the expense of each function evaluation in problem 14 by a factor of 100. The cost of a function evaluation is then about the same order of magnitude as the cost of a function evaluation in problem 10. For $p = 16$ and $n = 100$ the running time is 177.4 compared to 1562.3 when $p = 1$; the resultant speedup is 8.9, which compares favorably to the speedups obtained on problems 9 and 10.

The purpose of Table 4 is to provide some indication of how the computing time is distributed amongst the various tasks. The table entries represent the normalized time to do each task—for each row, each task time is divided by the time required by the most expensive task. Column "Int. J/QR" represents the time to do the initial estimation of J by finite-differences plus the time required to do the initial QR-factorization. The "F-evaluation" column represents the total time spent on evaluations of F excluding the initial estimation of J. The column labeled "Other" reflects the time spent on all remaining tasks. This includes several parallel matrix multiplies (used in the determination of Cauchy and Newton steps) as well as some sequential work (such as the adjustment of Δ).

Problem 14+ is the Minpack problem 14 with the cost of a function evaluation increased by a factor of 100.

TABLE 4
Secant Algorithm breakdown for $p = 16$, $n = 100$.

Problem	Init. J/QR	QR-update	F-evaluation	Tri. solve	Other
9	1.0	.26	.07	.22	.18
10	1.0	.14	.06	.10	.10
14+	1.0	.28	.42	.28	.06
15	.10	1.0	.27	.56	.40

It is now easy to see why problem 15 experiences relatively poor speedup: too little (relative) time is spent on the highly parallel tasks such as the initialization step and "F-evaluation." Instead, the QR-update and the triangular solve tasks consume the most time: unfortunately, neither task is as parallel-efficient as the QR-factorization or the distributed evaluation of F.

Our numerical experiments indicate to us that the proposed parallel secant method is acceptably efficient (for both cheap and expensive functions) except when F is cheap *and* many iterations are required. Note that many iterations may be required due to a poor Jacobian approximation; a hybrid routine such as the Minpack Algorithm would tend to refresh the approximation (e.g., by finite differences) under such circumstances and this, in turn, would tend to decrease the number of iterations. In particular, we note that the parallel finite-difference method actually outperforms the parallel secant method on problems 14 and 15: this is due to many fewer iterations and (ridiculously) cheap function evaluations.

4. Parallel finite-difference Newton method.

4.1. The algorithm. In the remainder of this paper we assume that the evaluation of $F(x)$ is not a distributed computation; every node has a copy of the F-evaluation subroutine. In addition, we discontinue the use of the subscript c; e.g., x and s refer to vectors x_c and s_c, respectively. The finite-difference approximation of the Jacobian matrix can obviously be done in parallel, given $F(x)$, with each node computing its resident columns independently. Communication between nodes is not required. Since efficient parallel routines for the LU and QR factorizations exist, and since the triangular solve problem has been extensively researched, with acceptable results, the only remaining difficulty is the evaluation of $F(x+s)$ when determining if $x+s$ is an acceptable point. If F is relatively expensive to compute, then it is not reasonable to designate one distinguished node to evaluate $F(x+s)$ while the others idle.

Our solution breaks into two parts. First, if in the course of a run previous experience suggests that the initial trial step s is likely to be accepted, then we take a chance and overlap the computation of $F(x+s)$ with the estimation of $J(x+s)$. This is at some risk because s might be determined to be unacceptable—due to the value of $\|F(x+s)\|$—and then any work expended on the computation of $J(x_c+s)$ has been wasted. But, as indicated in Fig. 6, each node will waste at most one evaluation of F before the suitability of s is determined.

Assign the task of evaluating $F(x+s)$ to node $n \bmod p$ (think of F as column $n+1$ of the Jacobian): $n+1 \in I(n \bmod p)$. Figure 6 describes the algorithm.

```
If {$myid = n \bmod p$} then {i.e., if I am the node that evaluates $F(x+s)$}
    Evaluate $F(x+s)$;
    Broadcast $F(x+s)$;
Else
    Choose $j \in I(myid)$;
    Evaluate $F(x+s+\tau e_j)$;
Endif

If $x+s$ is not acceptable then exit
Else
    Evaluate the remainder of the Jacobian columns by finite differences;
Endif
```

FIG. 6. *Algorithm* J-*and*-F-*evaluation* (*node program*).

On the other hand, if previous experience suggests that there is a good chance that the initial step s will not be adequate, then our strategy is quite different. Specifically, each step of the parallel linesearch procedure involves p function evaluations, $F(w_{p-1}), \cdots, F(w_0)$, done in parallel, where $w_{p-1}, \cdots, w_0$ are points along the dogleg step P_c (see § 2). If the first step is unsuccessful, then a second parallel search is performed along a smaller segment of P_c. This process is repeated until a suitable point is found. The linesearch procedure is judged successful if the following "alpha condition" is satisfied for some $w_* \in \{w_{p-1}, \cdots, w_0\}$:

$$f(w_*) \leqq f(x) + \alpha \nabla f(x)^T [w_* - x], \tag{10}$$

where $0 < \alpha < \frac{1}{2}$, and $f(x) \overset{\text{def}}{=} \frac{1}{2}\|F(x)\|_2^2$. See Dennis and Schnabel [3] for a discussion of the "alpha condition."

The procedure can be considered a parallel *generalized* bisection algorithm or perhaps a generalized Armijo rule [1]. In each step we begin with a stepsize bound of Δ. Specifically, in the first step

$$\Delta \leftarrow \min \{\|s^{\text{Newton}}\|, \text{BOUND} \times \text{XNORM}\} \tag{11}$$

where $\text{XNORM} \overset{\text{def}}{=} \max \{\|x\|, \text{TYPX}\}$ and TYPX is a positive user-supplied constant representing the norm of a "typical" x-iterate; BOUND is a positive user-supplied constant.

In subsequent steps (if needed) Δ is defined by the (unsuccessful) evaluation point nearest to x (used in the previous step).

In each step the evaluation points are defined as follows:

$$w_i \in P_c, \quad \|w_i - x\| = \frac{\Delta}{\gamma^i}, \quad i = 0 \colon p-1 \tag{12}$$

where γ is a positive number strictly greater than unity. The choice of γ is guided by the following two concerns.

First, γ should be chosen so that the point nearest to x, w_{p-1}, is not too close to x: i.e., we require

$$\frac{\Delta}{\gamma^{p-1}} \geqq \mu \times \text{XNORM} \tag{13}$$

where μ is small positive number (usually unit roundoff). We assume $\text{BOUND} \gg \mu$. Obviously expression (13) yields the upper bound on γ,

$$\gamma \leqq \left(\frac{\Delta}{\mu \times \text{XNORM}} \right)^{1/(p-1)} \tag{14}$$

Second, it is usually advantageous to spread the evaluation points so that at least one point (i.e., w_{p-1}) is on the Cauchy segment $(x, x + s^{\text{Cauchy}}]$. This leads to the condition, if $\|s^{\text{Cauchy}}\| < \Delta$, we require

$$\frac{\Delta}{\gamma^{p-1}} \leqq \|s^{\text{Cauchy}}\|. \tag{15}$$

However, this condition may lead to a γ so large that a very large segment of $s^{\text{Newton}} - s^{\text{Cauchy}}$ is devoid of function evaluations. Hence we compromise (15) and require only

$$\gamma \geqq \min \left\{ \left(\frac{\Delta}{\|s^{\text{Cauchy}}\|} \right)^{1/(p-1)}, 2 \right\}. \tag{16}$$

Note that if we assume that

$$\|s^{\text{Cauchy}}\| > \mu \times \text{XNORM}, \tag{17}$$

then (14) and (16) are consistent and $\gamma > 1$. If condition (17) does not hold, then it is reasonable to stop, claiming optimality.

Figure 7 exhibits the algorithm. The "Decrease Δ" step is implemented as follows. Let $p^+ = p+1$; determine $\gamma^+ > 1$ satisfying (14) and (16), replacing p with p^+. Finally, assign

$$\Delta \leftarrow \frac{\Delta}{[\gamma^+]^{(p^+-1)}}. \tag{18}$$

The "Fan-in" step determines $w^* \in W = \{w_{p-1}, \cdots, w_0\}$ such that $\|w^* - x\|$ is maximum and w^* satisfies (10). If no such point exists we define $w^* = x$. Determining if (10) is satisfied at point w_i is a simple computation. Specifically, we can write

$$w_i - x = \lambda_i^C s^{\text{Cauchy}} + \lambda_i^N s^{\text{Newton}}. \tag{19}$$

But, $\nabla f^T s^{\text{Newton}} = -\|F(x)\|_2^2 \overset{\text{def}}{=} \omega^N$ and $\nabla f^T s^{\text{Cauchy}} = -\beta \|J^T F\| \overset{\text{def}}{=} \omega^C$. (The constant β is computed when P_c is determined.) Therefore,

$$\nabla f^T [\omega_i - x] = \lambda_i^C \omega^C + \lambda_i^N \omega^N, \tag{20}$$

which is a trivial expression to compute on a single node.

The overall procedure is sketched in Fig. 8. Note that we have provided a high-level global view, as opposed to a node program. The parallelization of each computationally significant step has been discussed above.

4.2. Numerical results. We performed computational experiments using problems 9, 10, 14, and 15 referred to previously. The standard starting point (Factor = 1) was used in all cases. The stopping criterion was $\|F\| \leqq 10^{-8}$.

Problem 10 is distinguished from the others: F is relatively expensive to compute. In this sense problem 10 probably represents a more realistic test function. However, problems 14 and 15 are useful for testing because nonunit steplengths are required, whereas Newton iterations converge quickly, with unit steps, for problems 9 and 10.

In Table 5 we have recorded the iteration counts and running times obtained for the finite-difference Newton method described above (e.g., y/z indicates y iterations taking a total of z seconds); in Table 6 we divide the $p = 1$ running times by the $p = 4$ and $p = 16$ times to obtain a measure of speedup. In this case we do *not* compare our algorithm to a finite-difference version of the Minpack code because we feel such a

```
Choose $\Delta$ as in 11
**Repeat**
    **If** {$myid = 0$} **then**
        Determine $\gamma > 1$ satisfying (14) and (16);
        Broadcast $\gamma$;
    **Endif**
    $i \leftarrow myid$;
    Determine $w_i \in P_c$: $\|w_i - x\| = \Delta/\gamma^i$;
    Evaluate $z_i \overset{\text{def}}{=} F(w_i)$;
    Participate in fan-in: $z \to z^*$, $w \to w^*$ on node 0;
    Decrease $\Delta$;
**Until** $f(w_*) \leqq f(x) + \alpha \nabla f(x)^T [w_* - x]$
```

FIG. 7. *Algorithm Dogleg-Linesearch (node program).*

```
Guess an initial $x$;
Evaluate $F(x)$, $J(x)$;
Assign prev-step ← "not-Newton";
Repeat
    Factor $J = LU$;
    Determine $P_c$:
        Compute $s^{\text{Cauchy}}$; {see Fig. 1}
        Compute $s^{\text{Dogleg}}$; {see Fig. 1}
        Determine $\Delta$; {use (11)}
    If {prev-step = "Newton"} then
        Try algorithm J-and-F;
        If {J-and-F is unsuccessful} then
            prev-step ← "not-Newton"
        Endif
    Endif
    If {prev-step = "not-Newton"} then
        Perform Dogleg-Linesearch;
        Evaluate $J(x^+)$;
    Endif
Until {convergence}
```

FIG. 8. *Algorithm Newton with Dogleg-Linesearch.*

TABLE 5
Results for the Newton Algorithm with Dogleg-Linesearch.

Problem	p	$n=50$	$n=100$	$n=300$
9	1	3/7.6	3/43.5	3/911.0
9	4	3/3.3	3/13.6	3/240.1
9	16	3/3.0	3/7.9	3/79.8
10	1	4/107.9	4/826.8	4/21701.4
10	4	4/30.4	4/220.0	4/5528.2
10	16	4/13.1	4/69.2	4/1447.4
14	1	9/41.1	9/218.2	9/4050.2
14	4	9/15.5	9/65.7	9/1062.6
14	16	9/12.3	9/33.4	9/344.0
15	1	22/65.0	27/496.3	31/12871.5
15	4	22/30.1	27/158.2	31/3398.8
15	16	11/13.7	12/40.6	14/496.1

comparison would be unfair. Specifically, the Minpack code is QR-based and ours is LU-based: such a comparison would give meaningless advantage to our method.

Our implementation struggles with problem 15 for small p (relative to its performance with $p = 16$). This is because our linesearch parameter γ was chosen with a moderately large p in mind. In particular, in a first iteration of the Linesearch Algorithm, we choose γ so that condition (16) is an equality. If a second iteration of the linesearch is needed, then γ is chosen to satisfy condition (14) exactly. This strategy appears to work quite well for $p = 16$ but can lead to small steps if p is small.

In Table 6 we have normalized the execution times reported in Table 5: for each problem divide the execution times in the subcolumn by the $p = 1$ execution time. Hence the entries in Table 6 reflect the speedup factor over the running time on a single node.

TABLE 6
Speedups for the Newton Algorithm with Dogleg-Linesearch.

Problem	p	$n=50$	$n=100$	$n=300$
9	4	2.3	3.2	3.8
9	16	2.5	5.5	11.4
10	4	3.5	3.8	3.9
10	16	8.2	11.9	15.0
14	4	2.7	3.3	3.8
14	16	2.5	5.5	11.4
15	4	2.2	3.1	3.8
15	16	4.7	12.2	25.9

Remarks on Table 6. (1) For fixed $p>1$, the speedup improves as n increases. The primary reason for this is that as n increases the parallel factorization becomes increasingly efficient (e.g., [9]).

As Table 7 indicates, the factorization accounts for a significant percentage of the total computational expense in the test problems.

(2) In general, the 4-processor speeds are closer to optimality (optimal speedup = 4) than the 16-processor speedups (optimal speedup = 16). Again, the primary factor here is the increased efficiency of the parallel LU-factorization as n/p increases. There are two exceptions to this trend in Table 6.

First, a nearly optimal speedup is obtained on problem 10, $n=300$. Table 7 explains this: the Jacobian estimation time dominates the factorization time—parallel finite-difference is a highly parallel task.

The other exception occurs on problem 15, $n=300$: a speedup of 25.9 is attained (considerably better than the "optimal" speedup of 16!). This is possible because the sequence of points generated is a function of the number of processors used (due to the linesearch). This dependence contrasts with the Parallel Secant Algorithm described in the previous section.

In Table 7 we break the total execution time down into the times required by the different substeps. Each row of Table 7 is normalized so that the maximum entry in each row is unity.

Except for problem 10, the factorization represents the dominant cost. We believe this is an anomaly: in practice function evaluations are often quite expensive. However, in either case, the linesearch and triangular solve times are relatively insignificant.

We are satisfied with the performance of this parallel finite-difference algorithm that combines a dogleg step with a generalized bisection algorithm. Our experience on our test collection indicates that the required number of iterations is almost always fewer than for a dogleg/trust region strategy (i.e., no linesearch). However, we admit that from an aesthetic point of view the linesearch procedure is unattractive: moreover,

TABLE 7
The Newton Algorithm breakdown for $p=16$, $n=300$.

Problem	Jac. est.	Factor	Linesearch	Tri. solve
9	.08	1.0	.01	.08
10	1.0	.11	.02	.01
14	.17	1.0	.00	.08
15	.04	1.0	.01	.08

it is somewhat heuristic in nature and is unrelated to the quadratic model philosophy. Considering these remarks, it might be preferable, overall, to stick with the usual dogleg/trust-region philosophy and always employ the algorithm in Fig. 6 to obtain parallelism. Our experiments indicate this would be slightly less efficient.

5. Parallel multiple secant method.

5.1. The algorithm. The obvious disadvantage to the finite-difference Newton method discussed in § 4 is that the estimation of the Jacobian matrix can be extremely time-consuming. This is less true in the parallel context since finite-differencing is a highly parallel task; nevertheless, it is often unnecessary to obtain such accuracy. The success of the sequential rank-1 secant method attests to this claim.

Section 3 presented a parallel secant method under the row-separability assumption; however, if F is to be treated as a single entity we do not know how to implement an efficient rank-1 secant method (when the evaluation of F is expensive). But, it is possible to fill the gap between rank-1 and rank-n (i.e., finite-difference approximation) with an efficient parallel rank-q secant method where q is a multiple of p, the number of nodes.

To introduce the *multisecant method* let us consider the case when $q=p$. For the moment we also assume that our algorithm is purely local: a unit step $x^+ \leftarrow x+s$ is always taken (we consider the general situation later).

Assume that the function value $F(x)$ is known to every node. The first step is to re-label the nodes so that a *ring* is induced (i.e., node i is a neighbor of both node $(i+1) \bmod p$ and node $(i-1) \bmod p$, for $i=0{:}\,p-1$). A gray code mapping can be used for this purpose (e.g., [10]). Assume that B_c is distributed in the usual fashion, using this labeling. Hence, node j is assigned column k provided $k-1=j \bmod p$. Let s be the correction to x: i.e., $x_+ \leftarrow x+s$. (We assume for the moment that s will be accepted.)

The next step is key. Each node evaluates F at a different point. Node 0 evaluates $F(x+s^0)$, where $s^0=s$; node j, $1 \leqq j \leqq p-1$, evaluates $F(x+s^j)$ where s^j is a sparse projection of s. That is, component i of s^j will be either s_i or zero. In particular,

$$\text{if } \{i-1=j \bmod p\} \text{ or } \{s_i^k=0,\ 0 \leqq k<j\} \text{ then } s_i^j=0, \tag{21}$$

$$\text{otherwise } s_i^j=s_i. \tag{22}$$

For example, if $p=q=4$, $n=8$,

$$s^0=(s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8), \tag{23}$$

$$s^1=(s_1, 0, s_3, s_4, s_5, 0, s_7, s_8), \tag{24}$$

$$s^2=(s_1, 0, 0, s_4, s_5, 0, 0, s_8), \tag{25}$$

$$s^3=(s_1, 0, 0, 0, s_5, 0, 0, 0). \tag{26}$$

After evaluation, each node sends a copy of its newly computed function value to its higher numbered neighbor on the ring. Hence, after this shift, node j will have the vectors $F(x)$, $F(x+s^j)$, and $F(x+s^{(j-1) \bmod p})$. Therefore, if $p=q=4$ then in addition to $F(x)$, each node has the pair of function values listed below:

$$\text{node } 0{:}\ F(x+s),\ F(x+s^3), \tag{27}$$

$$\text{node } 1{:}\ F(x+s^1),\ F(x+s), \tag{28}$$

$$\text{node } 2{:}\ F(x+s^2),\ F(x+s^1), \tag{29}$$

$$\text{node } 3{:}\ F(x+s^3),\ F(x+s^2). \tag{30}$$

We now demand that each node satisfy its own local secant equation.

Notation. For a matrix M let $M_{I(j)}$ denote the matrix of the same dimensions that matches M in columns $I(j)$ and whose other columns are zero columns. For $j=1\colon p-1$ the secant equation for node j is

$$B^+_{I(j)}[(x+s^{j-1})-(x+s^j)]=y^j \tag{31}$$

where $y^j \overset{\text{def}}{=} F(x+s^{j-1})-F(x+s^j)$. Equation (31) is reasonable because $(s^{j-1}-s^j)_i\neq 0 \Rightarrow i\in I(j)$ and

$$\left\{\int_0^1 J_{I(j)}([x+s^j]+\tau[s^{j-1}-s^j])\,d\tau\right\}(s^{j-1}-s^j)=y^j. \tag{32}$$

On node 0 we demand satisfaction of the secant equation

$$B^+_{I(0)}[s^{p-1}]=y^0 \tag{33}$$

where $y^0 \overset{\text{def}}{=} F(x+s^{p-1})-F(x)$.

This requirement is also reasonable because $s_i^{p-1}\neq 0\Rightarrow i\in I(0)$, and

$$\left\{\int_0^1 J_{I(0)}(x+\tau s^{p-1})\,d\tau\right\}s^{p-1}=y^0. \tag{34}$$

Of course "reasonableness" does not establish that the method possesses desirable local convergence properties. We will consider this theoretical question elsewhere [16].

An important property of this parallel multisecant update is that there is very little communication required: each node sends (receives) exactly one vector to (from) an *adjacent* node. Moreover, beyond satisfying p local secant equations, the updated matrix B^+ also satisfies the *global* secant equation

$$B^+s=y \tag{35}$$

where $y \overset{\text{def}}{=} F(x+s)-F(x)$. To see this note that

$$B^+_{I(0)}s^{p-1}+B^+_{I(1)}(s^0-s^1)+\cdots+B^+_{I(p-1)}(s^{p-2}-s^{p-1})=B^+s \tag{36}$$

and

$$y^0+\cdots+y^{p-1}=F(x+s)-F(x)=y. \tag{37}$$

Figure 9 summarizes the node program representing the algorithm sketched above.

```
$j \leftarrow myid$;
Evaluate $F(x+s^j)$;
Send a copy of the vector $F(x+s^j)$ to node $(j+1) \bmod p$;
Receive a copy of the vector $F(x+s^{(j-1) \bmod p})$ from node $(j-1) \bmod p$;

If {$myid=0$} then
    $y^0 \leftarrow F(x+s^{p-1})-F(x)$;
    $d^0 \leftarrow s^{p-1}$;
Else
    $y^j \leftarrow F(x+s^{j-1})-F(x+s^j)$;
    $d^j \leftarrow s^{j-1}-s^j$;
Endif

{Update resident columns}
If {$d^j \neq 0$} then

    $B^+_{I(j)} \leftarrow B_{I(j)} + \frac{(y^j-Bd^j)d^{j^T}}{d^{j^T}d^j}$;

Endif
```

FIG. 9. *Multiple secant update (node program).*

Note that the product Bd^j can be computed entirely locally on node j: the vector d^j has nonzero components only in locations corresponding to columns residing on node j (i.e., the nonzero columns in $B_{I(j)}$).

Two subtopics remain to be discussed: the generalization of the multisecant method to the case where q is a multiple of p, and the globalization strategy.

The generalization of the multisecant method to the case where q is a multiple of p is quite straightforward. Divide the columns on each node into q/p groups (typically with as many equal-sized groups as possible). Each group accounts for one evaluation of F; a local secant equation is defined with respect to each group. Otherwise, the method is the same as the $p=q$ case except now the ring is viewed as having q conceptual nodes. Note that the number of transferred messages between physical nodes remains at p (neighbor-to-neighbor, each message of size n).

Globalization can be achieved in a manner similar to the parallel finite-difference algorithm. Indeed, the only change is to replace the finite-difference Jacobian calculation with the local secant update. With this exception, the algorithm described in Fig. 8 can be used unaltered.[2] We note that if $p=q$ and Newton steps are being successfully used, then each node is involved in exactly one F-evaluation per iteration (in each Newton iteration there are p F-evaluations). However, when the secant update follows a linesearch there is a slight redundancy. Specifically, after the linesearch procedure is completed the value $F(x^+)$ is known. But the local secant update requires only $p-1$ additional F-evaluations beyond $F(x^+)$. Therefore, under these circumstances either one node remains idle during the parallel evaluation of F for the multiple secant update, or $F(x^+)$ is computed twice.

5.2. Numerical results. Experimentally we compared the multisecant method to the parallel finite-difference procedure discussed in § 4 using problems 9, 10, and 14 of the Minpack collection and the extended Rosenbrock function (problem 15). Table 8 provides the results. (Table entry y/z indicates y iterations taking a total of z seconds.)

TABLE 8
The multisecant method versus the finite-difference Newton method, $p=16$.

Problem	Method	$n=50$	$n=100$	$n=300$
9	MS	3/4.6	3/12.3	3/140.9
9	FD	3/3.0	3/7.9	3/79.8
10	MS	4/12.4	4/46.6	4/733.6
10	FD	4/13.1	4/69.2	4/1447.4
14	MS	20/32.4	23/99.3	23/1175.8
14	FD	9/12.3	9/33.4	9/344.0
15	MS	21/35.2	25/101.9	41/1912.6
15	FD	11/13.7	12/40.6	14/496.1

In general, the secant method requires more iterations. Therefore, when the factorization is the dominant cost—as it is in problems 9, 14, and 15—then the parallel finite-difference Newton method is faster. However, when F is expensive—as it is in problem 10—the multisecant method is probably preferable. Indeed, we have tried

[2] The remarks concerning linesearch versus trust region, made at the end of the previous section, are applicable to the globalized multisecant method as well.

problem 14+ (i.e., increase the expense of a function evaluation in problem 14 by a factor of 100) with $n = 100$: the multisecant then requires 405.5 seconds compared to 551.1 required by the parallel finite-difference method.

To determine where the algorithm spends most of its time, on this test collection, Table 9 provides a breakdown of the total execution time.

Table 9 is fairly similar to Table 7; however, the factorization cost for problem 10 is relatively more expensive (.38 versus .11). This is because the dominant cost—Jacobian estimation—has decreased considerably. In a similar vein, the relative Jacobian costs have almost become insignificant in problems 9, 10, and 15.

As mentioned above, the multisecant method can allow for the independent estimation of several groups per node (as opposed to just one group per node). In Table 10 we provide results indicating the effect of varying the number of groups, q, per node.

The results are as expected. For problem 10 the computational expense increases as q increases. This is because of the expensive nature of F (and the number of iterations stays constant). Problem 15 exhibits exactly the opposite behavior: as q increases the execution time decreases. The reason for this is that the number of iterations decreases as q increases: this is reasonable since the Jacobian approximations become increasingly accurate as q increases. In general then, the optimal q will depend on the particular problem: the relative costs of evaluating F, factoring the matrix, and the convergence dependence on q, play a role.

6. Conclusions.

We have proposed parallel algorithms for the solution of systems of nonlinear equations $F(x) = 0$. The algorithms are applicable on local-memory multiprocessors (such as a hypercube computer) provided that each processor has significant memory and computational power and the problem dimension is greater than the number of processors.

TABLE 9
The Multisecant Algorithm breakdown for $p = 16$, $n = 300$.

Problem	Jac. est.	Factor	Linesearch	Tri. solve
9	.04	1.0	.00	.07
10	1.0	.38	.04	.02
14	.05	1.0	.00	.06
15	.02	1.0	.01	.07

TABLE 10
Multisecant: Vary # groups-per-node, $p = 16$, $n = 300$.

Problem	# groups-per-node	# iterations	Total time
10	1	4	733.6
10	2	4	804.7
10	4	4	932.4
10	8	4	1087.2
15	1	41	1912.6
15	2	40	1679.8
15	4	29	1003.6
15	8	21	853.6

When F is efficiently computable in a distributed parallel manner, the globalized rank-1 secant method can be efficiently parallelized. Specifically, it is possible to parallelize the Minpack implementation, updating the QR factorization of an approximate Jacobian matrix every step. On balance we are satisfied with the parallel performance of our global parallel secant implementation. However, it should be noted that we have efficiently distributed and parallelized the F-subroutines in our experiments. In general such a task falls to the user and speedup will be strongly affected by the user's success in this task. Note that distributing F by rows, even if feasible, does not always lead to the most efficient distribution of work since it does not exploit the presence of common expensive subexpressions.

Since it is not always possible to efficiently parallelize the computation of F, we have developed parallel finite-difference and multisecant methods. In general it is difficult to achieve good speedup, relative to the sequential rank-1 secant method, for this class of functions; however, the finite-difference algorithm is efficiently parallelized and the multisecant method will generally improve on this (especially for large n/p). It is perhaps possible to further improve on the efficiency of the multisecant method by incorporating parallel multirank updates to the current distributed QR factorization. We have not yet investigated this possibility.

Finally, we note that sparse systems are partially separable (i.e., each component function depends only on a few variables); therefore, in theory, it is usually possible to effectively evaluate $F(x)$ in a distributed parallel manner. Hence, it may be possible to efficiently solve large sparse systems of nonlinear equations using a parallel sparse secant method.

Acknowledgments. Our numerical experiments were performed with the assistance of the Advanced Computing Facility at the Cornell Center for Theory and Simulation in Science and Engineering, which is supported by the National Science Foundation and New York State. Specifically, we are grateful to Ashley Doershug for her help with the Intel iPSC.

REFERENCES

[1] L. Armijo, *Minimization of functions having Lipschitz continuous first partial derivatives*, Pacific J. Math., 16 (1966), pp. 1-3.

[2] C. Broyden, *A class of methods for solving nonlinear simultaneous equations*, Math. Comp., 19 (1965), pp. 577-593.

[3] J. E. Dennis and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[4] P. Gill and W. Murray, *Quasi-Newton methods for unconstrained optimization*, J. Inst. Maths. Appl., 9 (1972), pp. 91-108.

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

[6] M. Heath and C. Romine, *Parallel solution of triangular systems on distributed-memory multiprocessors*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 558-588.

[7] G. Li and T. F. Coleman, *A parallel triangular solver for a distributed memory multiprocessor*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 485-502.

[8] ———, *A new method for solving triangular systems on a distributed memory message-passing multiprocessor*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 382-396.

[9] C. Moler, *Matrix computations on distributed memory multiprocessors*, Tech. Report, Intel Scientific Computers, Beaverton, OR, 1987.

[10] C. Moler and D. S. Scott, *Communications utilities for the ipsc*, Tech. Report, Intel Scientific Computers, Beaverton, OR, 1986.

[11] J. J. Moré, B. S. Garbow, and K. E. Hillstrom, *User guide for minpack-*1, Tech. Report ANL-80-74, Argonne National Laboratory, Argonne, IL, 1980.

[12] M. J. D. Powell, *A hybrid method for nonlinear equations*, in Numerical Methods for Nonlinear Algebraic Equations, P. Rabinowitz, ed., Academic Press, New York, 1970, pp. 87-114.

[13] C. Romine and J. Ortega, *Parallel solution of triangular systems of equations*, Tech. Report RM-86-05, Department of Applied Mathematics, University of Virginia, Charlottesville, VA, 1986.

[14] Y. Saad and M. H. Schultz, *Topological properties of hypercubes*, Tech. Report CS-RR-389, Computer Science Department, Yale University, New Haven, CT, 1985.

[15] P. Wiley, *A parallel architecture comes of age at last*, IEEE Spectrum (1987), pp. 46-50.

[16] T. F. Coleman and G. Li, *Local convergence of the multi-secant method for the parallel solution of systems of nonlinear equations*, Appl. Math. Lett., 1 (1988), pp. 141-145.

SIAM J. Sci. Stat. Comput.
Vol. 11, No. 6, pp. 1136-1158, November 1990

A NEW MODIFIED CHOLESKY FACTORIZATION*

ROBERT B. SCHNABEL† AND ELIZABETH ESKOW†

Abstract. The modified Cholesky factorization of Gill and Murray plays an important role in optimization algorithms. Given a symmetric but not necessarily positive-definite matrix A, it computes a Cholesky factorization of $A+E$, where $E=0$ if A is safely positive-definite, and E is a diagonal matrix chosen to make $A+E$ positive-definite otherwise. The factorization costs only a small multiple of n^2 operations more than the standard Cholesky factorization. A new algorithm that has these same properties, but for which the theoretical bound on $\|E\|_\infty$ is substantially smaller, is presented. It is based upon two new techniques, the use of Gerschgorin bounds in selecting the elements of E, and a new way of monitoring positive definiteness. In extensive computational tests on indefinite matrices, the new factorization virtually always produces smaller values of $\|E\|_\infty$ than the existing method, without impairing the conditioning of $A+E$. In some cases the improvements are substantial. The new factorization has already been useful in optimization algorithms.

Key words. Cholesky factorization, optimization, nonpositive definite, Gerschgorin bounds

AMS(MOS) subject classifications. 65F30, 65K10, 15A23

1. Introduction.

The modified Cholesky factorization was introduced by Gill and Murray [2], and subsequently refined by Gill, Murray, and Wright [3] (hereafter referred to as GMW81). Given a symmetric, not necessarily positive-definite matrix $A \in R^{n \times n}$, it calculates a Cholesky (i.e., LL^T, or equivalently LDL^T) factorization of $A+E$, where E is zero if A is safely positive-definite, and E is a nonnegative diagonal matrix for which $A+E$ is positive-definite otherwise. When A is not positive-definite, there is an a priori error bound on how large E can be as a function of A; the practical intent is that E not be much larger than is necessary to make $A+E$ positive-definite. The factorization uses only about $n^2/2$ more operations than the normal Cholesky factorization, which costs approximately $n^3/6$ each multiplications and additions.

The modified Cholesky factorization has become very important in optimization algorithms. Its primary use is in line search methods for unconstrained optimization, where it is used to generate a descent search direction when the Hessian matrix is not positive-definite (see, e.g., GMW81). It is also used in line search methods for constrained optimization problems (GMW81), and in some trust region methods [1].

This paper presents a new modified Cholesky factorization algorithm that is intended for the same purposes as the current method. The new algorithm still costs only a small multiple of n^2 operations more than the standard Cholesky factorization. It possesses a much smaller a priori bound on the size of the diagonal matrix E, and in extensive computational tests, $\|E\|_\infty$ almost never is larger, and in many cases is considerably smaller, than that generated by the algorithm of GMW81. In fact, when A is not positive-definite, $\|E\|_\infty$ is usually close enough to the negative of the smallest eigenvalue of A that the new algorithm may be a useful, inexpensive way to estimate this eigenvalue.

The remainder of this paper is organized as follows. Section 2 contains a brief summary of the motivation and uses for the modified Cholesky factorization in optimization algorithms. Section 3 summarizes the goals of this factorization and the basic challenges that it presents, and § 4 briefly describes the GMW81 algorithm.

* Received by the editors January 16, 1989; accepted for publication (in revised form) October 13, 1989. This research was supported by Army Research Office grant DAAL-03-88-0086, National Science Foundation grant CCR-8702403, and National Science Foundation cooperative agreement DCR-8420944.

† Department of Computer Science, Campus Box 430, University of Colorado, Boulder, Colorado 80309.

In § 5 we present the new algorithm. It contains two main novel features, the use of Gerschgorin bounds in determining both the pivot sequence and the elements of E, and a new two-phase strategy for determining when a matrix is not positive-definite and needs to be perturbed. In § 6 we present the results of an extensive computational comparison of the behavior of the new and old factorizations on indefinite test matrices of dimensions 25 to 75. Section 7 contains some brief conclusions.

Throughout the paper we consider the Cholesky factorization, i.e., the factorization into LL^T, where L is lower triangular, as opposed to the LDL^T factorization, where L is unit lower triangular (ones on the diagonal) and D is a positive-diagonal matrix. The conclusions of the paper are true for either factorization. We use the Cholesky factorization because we believe it makes the exposition simpler. We use the version of the Cholesky factorization that makes a rank-one change to the remaining submatrix at each iteration (analogous to Gaussian elimination), rather than the version that delays the changes to any element until it is in the pivot column (analogous to Crout reduction). The use of the first version will be seen in § 5 to be important to our algorithm.

2. The use of the modified Cholesky factorization in optimization algorithms. The modified Cholesky factorization was introduced by Gill and Murray [2] in the context of a line search method for solving the unconstrained optimization problem

$$\underset{x\in R^n}{\text{minimize}}\, f: R^n \to R.$$

Unconstrained optimization methods generally base each iteration upon the quadratic model of $f(x)$ around the current iterate x_c

$$m(x_c+d)=f(x_c)+\nabla f(x_c)^T d+\tfrac{1}{2}d^T H_c d, \tag{2.1}$$

where H_c is the Hessian matrix $\nabla^2 f(x_c)$ or a symmetric approximation to it. If H_c is positive-definite, then the step $d_c=-H_c^{-1}\nabla f(x_c)$ is the minimizer of (2.1) and also a descent direction for $f(x)$, so that a satisfactory next iterate x_+ always can be found by choosing $x_+=x_c+\lambda_c d_c$ for some $\lambda_c>0$. If H_c has one or more negative eigenvalues, however, then the model (2.1) is unbounded below, and H_c may be singular or the direction $d_c=-H_c^{-1}\nabla f(x_c)$ may or may not be a descent direction for $f(x)$. In this case, Gill and Murray [2] suggested calculating $d_c=-(H_c+E_c)^{-1}\nabla f(x_c)$ as the search direction, where h_c+E_c is positive-definite, and again choosing $x_+=x_c+\lambda_c d_c$ for some $\lambda_c>0$ by a line search procedure. By standard convergence results, if $\|H_c\|$ is uniformly bounded above, $\|E_c\|$ is bounded above as a function of $\|H_c\|$, and the condition number of H_c+E_c is uniformly bounded above, then the sequence of iterates generated by a standard line search method that uses such search directions will be globally convergent in the sense that the limit of the sequence of gradients converges to zero. If $E_c=0$ when H_c is positive-definite, and $H_c=\nabla^2 f(x_c)$, then the method will also be quadratically convergent in the neighborhood of a strong local minimizer. (See Dennis and Schnabel [1] for a summary of these results.)

The algorithm of Gill and Murray [2] for choosing E_c satisfies all the aforementioned conditions on E_c. It also is very efficient in that it calculates either the Cholesky factorization of H_c if it is positive-definite, or the Cholesky factorization of H_c+E_c otherwise, at barely a higher total cost than a standard Cholesky factorization, without knowing a priori whether H_c is positive-definite or not. For these reasons, it has become a standard technique in line search methods for unconstrained optimization problems. A refined version of the algorithm that has performed very well is given in GMW81.

The modified Cholesky factorization is also used in some line search methods for solving constrained optimization problems (see GMW81) and in some trust region methods for optimization (see Dennis and Schnabel [1]). Schultz, Schnabel, and Byrd [4] show how to construct efficient and globally convergent trust region methods if a satisfactory lower bound on the most negative eigenvalue λ_1 of H_c is available. The methods described in this paper produce bounds that are satisfactory in this sense. We briefly discuss another possible use of our modified Cholesky factorization in trust region methods in § 7.

3. Goals and challenges of the modified Cholesky factorization. Given a matrix $A \in R^{n \times n}$ that is symmetric but not necessarily positive definite, the objective of the modified Cholesky factorization is to construct a Cholesky (LL^T) factorization of a positive-definite matrix $A+E$, where E is a nonnegative diagonal matrix. More specifically, the factorization has the following four goals: (1) If A is safely positive-definite, E should equal zero; (2) If A is indefinite, $\|E\|_\infty$ should not be much greater than $-\lambda_1(A)$, where $\lambda_1(A)$ is the most negative eigenvalue of A; (3) $A+E$ should be a reasonably well conditioned matrix; and (4) The cost of the factorization should only be a small multiple of n^2 operations more than the cost of the normal Cholesky algorithm.

One obvious way to select E would be to find $\lambda_1(A)$, and, if $\lambda_1(A) < 0$, let E equal $[-\lambda_1(A)+\varepsilon]I$, for some small positive ε. This would satisfy the first three goals, but the expense of finding the eigenvalues of a matrix exceeds the cost requirements specified in our final goal by at least an order of magnitude. Thus the major challenge in developing a modified Cholesky factorization is to satisfy the first three goals while not increasing the cost by more than $O(n^2)$. Among other things, this implies that a one-pass algorithm is essential.

There is a basic trade-off in deciding upon the size of each of the diagonal elements of the matrix E, as we now explain. Let the $n+1-j \times n+1-j$ principal submatrix remaining to be factored at the jth iteration, consisting of the current elements in rows and columns j through n, be denoted

$$A_j = \begin{bmatrix} \alpha_j & a_j^T \\ a_j & \hat{A}_j \end{bmatrix},$$

where $\alpha_j \in R$ is the current jth diagonal element, $a_j \in R^{n-j}$ is the current vector of elements in column j below the diagonal, and $\hat{A}_j \in R^{(n-j)\times(n-j)}$. (We will use the conventions that the subscripts of the elements in the vector a_j are $i=j+1$ through n, so that $(a_j)_i = A_{ij}$, $i=j+1, \cdots, n$, and that $A_1 = A$.) Then at the jth iteration, the normal Cholesky factorization algorithm computes $L_{jj} = \sqrt{\alpha_j}$, $L_{ij} = (a_j)_i / L_{jj}$, $i = j+1, \cdots, n$, and (assuming the changes to the remaining elements are not deferred)

$$A_{j+1} = \hat{A}_j - \frac{a_j a_j^T}{\alpha_j}.$$

In the modified Cholesky factorization, the computations are instead $L_{jj} = \sqrt{\alpha_j + \delta_j}$, $L_{ij} = (a_j)_i / L_{jj}$, $i = j+1, \cdots, n$, and

$$A_{j+1} = \hat{A}_j - \frac{a_j a_j^T}{\alpha_j + \delta_j},$$

where δ_j is greater than or equal to zero and is the jth diagonal element of the matrix E. The tradeoff between making δ_j large or small leads to the following dilemma. If α_j is negative and δ_j is chosen so small that $\alpha_j + \delta_j$ is barely greater than zero, then

$a_j a_j^T/(\alpha_j+\delta_j)$ will be large, and A_{j+1} will have large negative eigenvalues, implying that the elements of E in some remaining iterations will need to be large. On the other hand, if δ_j is large, then we have already added a large amount to the diagonal. The challenge lies in adding the appropriate amount to the diagonal of A at the appropriate time in the algorithm. This requires that the algorithm consider more information than just the value of α_j in choosing δ_j. It will be seen in §§ 4 and 5 that considering the values of a_j as well as α_j is sufficient to produce effective modified Cholesky factorization algorithms in both theory and practice.

4. The modified Cholesky factorization of Gill, Murray, and Wright. GMW81 give a modified Cholesky factorization algorithm that is designed to satisfy the four goals stated at the start of § 3. Given a symmetric but not necessarily positive-definite matrix $A \in R^{n\times n}$, it computes an LDL^T factorization of a matrix $A+E$, where E is a nonnegative diagonal matrix. In this section, we briefly review their method. To be consistent with the remainder of the paper, we restate their algorithm in terms of the Cholesky (LL^T) decomposition. This does not change any of the important properties of the algorithm that we discuss.

At each iteration, the algorithm of GMW81 first selects the maximum (in absolute value) diagonal element in the remaining principal submatrix A_j, and pivots it to the top left position by interchanging its row and column with the pivot (jth) row and column, respectively. Then, if A_j is now the permuted principal submatrix, with

$$A_j=\begin{bmatrix}\alpha_j & a_j^T\\ a_j & \hat{A}_j\end{bmatrix}, \tag{4.1}$$

where α_j is the diagonal element in the pivot column and a_j is the remainder of the pivot column, the elements of the next principal submatrix A_{j+1} are computed by

$$A_{j+1}=\hat{A}_j-\frac{a_j a_j^T}{\alpha_j+\delta_j}. \tag{4.2}$$

The value of δ_j at each iteration is chosen to be the smallest nonnegative number such that

$$0\leqq\frac{\|a_j\|_\infty^2}{\alpha_j+\delta_j}\leqq\beta^2,$$

where $\beta>0$ is an a priori bound selected to minimize a worst-case bound on $\|E\|_\infty$. If $\alpha_j<0$ and this value of δ_j is less than $-2\alpha_j$, then $\delta_j=-2\alpha_j$ instead.

What remains to be described is the choice of β. Let $\xi=$ the maximum magnitude of the off-diagonal elements of the original matrix A, and $\gamma=$ the maximum magnitude of the diagonal elements of A. Gill and Murray [2] produce an error bound on $\|E\|_\infty$ as a function of β for their algorithm, and show that it is minimized when $\beta^2=\xi/\sqrt{n^2-1}$. For that choice of β,

$$\|E\|_\infty\leqq 2(\sqrt{n^2-1}+(n-1))\xi+2\gamma, \tag{4.3}$$

or roughly

$$\|E\|_\infty\leqq 4n\xi+2\gamma \tag{4.4}$$

for moderate to large n. However this choice of β may cause positive-definite matrices A to be perturbed, so the selection of β is adjusted in order to avoid this. Gill and Murray [2] also show that the choice $\beta\geqq\sqrt{\gamma}$ guarantees that $E=0$ for positive-definite A. Thus their algorithm assigns β^2 to be the maximum value of γ, $\xi/\sqrt{n^2-1}$, or machine epsilon. If $\gamma > \xi/\sqrt{n^2-1}$, the usual case, then the error bound for this adjusted β becomes

$$\|E\|_\infty \leqq (n^2+1)\gamma + 2(n-1)\xi + \xi^2/\gamma, \tag{4.5}$$

which is larger than (4.3).

The modified Cholesky factorization algorithm of GMW81 has proven to be an effective factorization in the context of optimization algorithms, and as will be seen in § 6, does quite a good job of fulfilling the four goals stated at the beginning of § 3. (The cost of the algorithm is approximately n^2 comparisons, and $O(n)$ arithmetic operations, more than the standard Cholesky factorization.) It should be noted that while the diagonal pivoting employed by the algorithm of GMW81 does not affect the analysis described above, it is very important to its good practical performance. In particular, on the test problems in § 6, we found that $\|E\|_\infty$ for the GMW81 algorithm was often several orders of magnitude smaller with pivoting than without it.

There appear to us to be two important ways in which the algorithm of GMW81 might still be improved. First, the bounds (4.3) and particularly (4.5), which are attained by the algorithm for particular matrices A, are far from optimal, as will be discussed in § 5. Second, the results of § 6 show that in practice, the value of $\|E\|_\infty$ produced by the algorithm is sometimes many times larger than necessary. The new method described in § 5 primarily attempts to improve upon the algorithm of GMW81 in these two regards.

5. The new modified Cholesky factorization. Our modified Cholesky factorization algorithm incorporates two new main techniques. The first involves using Gerschgorin circle theorem bounds to determine the elements in the nonnegative diagonal matrix E that is added to an indefinite matrix A in order to make it positive-definite. The second is a new technique for assuring that we do not perturb an already positive-definite matrix, i.e., that $E=0$ if A is positive-definite. In § 5.1 we describe the new technique that uses Gerschgorin bounds to decide how much to add to the diagonal, and show that it leads to an improved upper bound on $\|E\|_\infty$. In § 5.2 we describe the new technique for assuring that a positive-definite matrix is not perturbed, and show that unlike the strategy of GMW81, it can be incorporated into a modified Cholesky decomposition algorithm without causing the bound on $\|E\|_\infty$ to grow significantly. In § 5.3 we describe our full new algorithm, which integrates these two techniques, discuss its theoretical properties, and give a simple example comparing it to the method of GMW81.

5.1. Using Gerschgorin circle theorem bounds to determine the amounts to add to the diagonal. In this section, we introduce our basic strategy for choosing a nonnegative diagonal matrix E such that $A+E$ is positive semidefinite. (The exposition and theory are cleaner if we allow for the possibility that $A+E$ is positive *semi*definite; the changes to assure that it is strictly positive-definite are small in practice and theory, and are described in § 5.3.) The strategy described in this section may result in E having some positive elements even if A is positive-definite; the modifications we make to avoid this are described in § 5.2.

The Gerschgorin circle theorem (GCT) states that if $A \in R^{n\times n}$ is a symmetric matrix with eigenvalues $\lambda_1 \leqq \cdots \leqq \lambda_n$, then each $\lambda_i \in \{G_1 \cup G_2 \cup \cdots \cup G_n\}$, where

$$G_i = \left[A_{ii} - \sum_{\substack{j=1\\ j\neq i}}^{n} |A_{ij}|,\ A_{ii} + \sum_{\substack{j=1\\ j\neq i}}^{n} |A_{ij}|\right] \triangleq [G\,\mathrm{low}_i, G\,\mathrm{up}_i], \qquad i=1,\cdots,n. \tag{5.1.1}$$

Thus, since $A-\lambda_1 I$ is positive semidefinite, an upper bound on the amount that must be added to the diagonal of A to make $A+E$ positive semidefinite is

$$\text{Maxadd}_{\text{GCT}} \triangleq \max\{0, \max_i\{-G\,\text{low}_i\}\}. \tag{5.1.2}$$

An objective of the new modified Cholesky factorization is to find E for which $A+E$ is positive semidefinite and for which we can guarantee

$$\|E\|_\infty \leqq \text{Maxadd}_{\text{GCT}}, \tag{5.1.3}$$

at least in the case when we are not concerned about perturbing a positive-definite matrix. This bound is easily achieved as indicated by the following lemma and theorem. Note that since, using the notation of § 4,

$$\text{Maxadd}_{\text{GCT}} \leqq \gamma + (n-1)\xi, \tag{5.1.4}$$

(5.1.3) is guaranteed to be stronger than (4.3).

LEMMA 5.1.1. *Let $A \in R^{n\times n}$ have the Gerschgorin circle theorem bounds G_i, $i = 1, \cdots, n$ given in* (5.1.1). *Denote $A = \left[\begin{smallmatrix} \alpha & a^T \\ a & \hat{A}\end{smallmatrix}\right]$, where $\alpha \in R$, $a \in R^{n-1}$, $\hat{A} \in R^{(n-1)\times(n-1)}$. Let $\bar{A} = \hat{A} - aa^T/(\alpha+\delta)$ have Gerschgorin circle theorem bounds $\bar{G}_i$, $i = 2, \cdots, n$, where*

$$\bar{G}_i = \left[\bar{A}_{ii} - \sum_{\substack{j=2\\ j\neq i}}^{n} |\bar{A}_{ij}|,\ \bar{A}_{ii} + \sum_{\substack{j=2\\ j\neq i}}^{n} |\bar{A}_{ij}|\right] \triangleq [\bar{G}\,\text{low}_i, \bar{G}\,\text{up}_i], \qquad i = 2, \cdots, n.$$

Then if

$$\delta \geqq \max\{0, \|a\|_1 - \alpha\}, \tag{5.1.5}$$

$\bar{G}_i \subseteq G_i$, $i = 2, \cdots, n$.

Proof. Note that (5.1.5) guarantees $\alpha + \delta \geqq 0$, with equality possible only if $a = 0$. If $a = 0$, we may assume that we set $\bar{A} = A$ so that the lemma is trivially true. For the remainder of the proof, we assume $\alpha + \delta > 0$.

Let us again use the convention that the subscripts of the vector a are $i = 2$ through n, so that $a_i = A_{i1}$, $i = 2, \cdots, n$. Then we have

$$\text{row } i \text{ of } \bar{A} = \text{row } i \text{ of } A - \frac{a^T a_i}{\alpha+\delta}, \qquad i = 2, \cdots, n.$$

Thus

$$\left|\sum_{\substack{j=2\\ j\neq i}}^{n} |\bar{A}_{ij}| - \sum_{\substack{j=2\\ j\neq i}}^{n} |A_{ij}|\right| \leqq \frac{(\|a\|_1 - |a_i|)|a_i|}{\alpha+\delta}. \tag{5.16}$$

Also,

$$\bar{A}_{ii} - A_{ii} = -\frac{a_i^2}{\alpha+\delta}. \tag{5.1.7}$$

Combining (5.1.6) and (5.1.7), recalling that the term $A_{i1} = a_i$ is present in G_i but not in $\bar{G}_i$, and using $\delta \geqq \|a\|_1 - \alpha = -G\,\text{low}_1$, we get

$$\begin{aligned} \bar{G}\,\text{low}_i - G\,\text{low}_i &\geqq |a_i| - \frac{\|a\|_1 |a_i|}{\alpha+\delta} \\ &= \frac{|a_i|}{\alpha+\delta}(\delta + (\alpha - \|a\|_1)) \\ &= \frac{|a_i|}{\alpha+\delta}(\delta + G\,\text{low}_1) \geqq 0, \end{aligned} \tag{5.1.8}$$

$i=2,\cdots,n$. Similar calculations show that

$$\bar{G}\,\mathrm{up}_i - G\,\mathrm{up}_i \leqq -\frac{|a_i|}{\alpha+\delta}(\delta + G\,\mathrm{low}_1 + 2|a_i|) \leqq 0.$$

Thus $\bar{G}_i \subseteq G_i$. □

Lemma 5.1.1 shows that the choice (5.1.5) causes the Gerschgorin intervals to contract. Thus it is almost immediate that if we make this choice with equality at each iteration of the modified Cholesky factorization, we will satisfy (5.1.3).

THEOREM 5.1.2. *Let $A \in R^{n\times n}$ have the Gerschgorin circle theorem bounds* (5.1.1), *and let* $\mathrm{Maxadd}_{\mathrm{GCT}}$ *be defined by* (5.1.2). *Suppose that at each iteration of the modified Cholesky factorization, the remaining principal submatrix $A_j \in R^{(n+1-j)\times(n+1-j)}$ is given by* (4.1), $(A_1 = A)$,

$$\delta_j = \max\{0, \|a_j\|_1 - \alpha_j\}, \tag{5.1.9}$$

and $A_{j+1} \in R^{(n-j)\times(n-j)}$ is calculated by (4.2). *Let $E = \mathrm{diag}\{\delta_1,\cdots,\delta_n\}$. Then $A+E$ is positive semidefinite and* (5.1.3) *is true. Furthermore, if any diagonal pivoting strategy is used at each iteration* (*i.e., rows and columns i and j are swapped for some $i>j$*), (5.1.3) *remains true.*

Proof. The proof is almost immediate from Lemma 5.1.1. Let $(G^j)_i$, $i=j,\cdots,n$ denote the Gerschgorin interval obtained from row i of A_j, and let $(G^j\,\mathrm{low})_i$ denote the lower bound of $(G^j)_i$. From Lemma 5.1.1, the choice (5.1.9) assures that

$$(G^{j+1}\,\mathrm{low})_i \subseteq (G^j\,\mathrm{low})_i, \qquad 1 \leqq j \leqq i \leqq n. \tag{5.1.10}$$

From (5.1.9), (5.1.10), and (5.1.2),

$$\delta_j \leqq -(G^j\,\mathrm{low})_j \leqq -G\,\mathrm{low}_j \leqq \mathrm{Maxadd}_{\mathrm{GCT}}.$$

This completes the proof of the first part of the theorem. Since diagonal pivoting of a symmetric matrix only permutes its Gerschgorin intervals but does not alter them, and since Lemma 5.1.1 and the above part of this proof make no use of the ordering of the Gerschgorin intervals, the theorem is unaffected by any diagonal pivoting strategy. □

Our algorithm makes one further modification to the strategy (5.1.9) for selecting δ_j. It is that we require the amount that is added to the diagonal at iteration j to be at least as great as the greatest amount that has been added to the diagonal at any previous iteration. That is,

$$\delta_j = \max\{0, \|a_j\|_1 - \alpha_j, \delta_{j-1}\}. \tag{5.1.11}$$

It is straightforward that Theorem 5.1.2 remains true with (5.1.11) in place of (5.1.9), because by induction this choice still satisfies (5.1.3), and trivially it still satisfies (5.1.5).

The rationale for this modification is as follows. At any iteration, suppose δ_j given by (5.1.11) is larger than that given by (5.1.9), i.e., $\max\{0, \|a_j\|_1 - \alpha_j\} < \delta_{j-1}$. Then the new choice (5.1.11) does not change the value of $\|E\|_\infty$ at this point in the algorithm, because $\delta_j = \delta_{j-1}$. It may cause subsequent values of δ_i to be smaller, however, because it results in a larger $\alpha_j + \delta_j$ and hence a smaller multiple of $a_j a_j^T$ is subtracted from $\hat{A}_j$, which means that A_{j+1} has larger or identical eigenvalues than it would have using (5.1.9). This reasoning does not imply that the final value of $\|E\|_\infty$ will be smaller using (5.1.11) than using (5.1.9), but it makes this seem likely, and in practice the modification appears to be helpful in some cases and virtually never harmful.

The total additional work required by the modifications to the Cholesky factorization described so far in this section is approximately $n^2/2$ additions, for the computation of $\|a_j\|_1$ at each iteration. In comparison, the additional work for the algorithm of GMW81 is approximately $n^2/2$ comparisons, because it computes $\|a_j\|_\infty$.

Finally, as noted in § 4, it is important in practice to use a diagonal pivoting strategy, even though it does not affect the theoretical results given above. We could simply pivot based on the maximum diagonal element, as is done by GMW81. However, recall that the amount we add to the diagonal at iteration j will be at least the negative of the lower Gerschgorin bound of the pivot row for that iteration. This suggests that we instead select as pivot row (and column) the row (and column) for which the lower limit of the Gerschgorin interval is largest. If this Gerschgorin bound is positive, then we will not increase $\|E\|_\infty$ at this iteration, and the Gerschgorin intervals will contract.

This pivoting strategy assumes that the Gerschgorin bounds for each remaining row are available at each iteration. This would require a total of approximately $n^3/2$ additional additions, which is too high. An alternative is to pivot based on the estimates of the Gerschgorin bounds that result from the proof of Lemma 5.1.1. If we let $(g^j)_i$ denote the estimate of the lower bound of the Gerschgorin interval of row i of A_j, then from (5.1.8),

$$(g^{j+1})_i = (g^j)_i + |(a_j)_i| \left[1 - \frac{\|a_j\|_1}{\alpha_j + \delta_j} \right], \qquad i = j+1, \cdots, n.$$

For the entire algorithm, this requires approximately $n^2/2$ each additional multiplications and additions. To begin this process, the Gerschgorin bounds of the original matrix A must be calculated, which costs an additional n^2 additions. Thus the total costs of the modifications to the Cholesky factorization discussed in the section are $2n^2$ additions and $n^2/2$ multiplications. The approximate Gerschgorin bounds calculated by this strategy may be quite inexact, but they are only used to determine pivot selection, and as we will see in § 6, substituting them for the exact Gerschgorin bounds does not significantly affect the performance of the algorithm.

We should mention that the strategy for preserving positive definiteness that we discuss in § 5.2 will often cause the additional costs given in this section to be reduced considerably.

5.2. The strategy for not perturbing positive-definite matrices. In this section we introduce our strategy for assuring that our modified Cholesky decomposition does not perturb an already positive-definite matrix, while still guaranteeing that if the matrix is not positive-definite, then the amount that is added to the diagonal is not too large. The strategy is quite simple. We divide our decomposition algorithm into two phases. In the first phase, we apply the standard Cholesky decomposition (the version described in § 3 where we make a rank-one modification to the remaining submatrix at each iteration) for $k \geqq 0$ iterations, stopping at the first occasion that the next, $(k+1)$st iteration would cause any diagonal element in the next remaining submatrix A_{k+2} to become nonpositive. At this point we know that the current submatrix A_{k+1}, as well as the original matrix A, is not positive-definite. We then switch to the second phase, where we apply the modified Cholesky decomposition algorithm described in § 5.1 for the remaining $n-k$ iterations of the decomposition.

If the original matrix A is numerically positive-definite, then this strategy results in the normal Cholesky decomposition being performed throughout. If A is not positive-definite, then this strategy results in the normal Cholesky decomposition being performed for $k \in [0, n-2]$ iterations, followed by the application of the modified

Cholesky decomposition to A_{k+1}, which results in the Cholesky decomposition of $A_{k+1}+\hat{E}$ for some nonnegative diagonal matrix $\hat{E}$. The overall result is the Cholesky decomposition of $A+E$, where E is $\hat{E}$ augmented with zeros in the first k diagonal positions (modulo pivoting).

The crucial question is "how large is $\|\hat{E}\|_\infty$, and hence $\|E\|_\infty$?." Section 5.1 gives a bound for $\|\hat{E}\|_\infty$ that depends on the sizes of the elements of A_{k+1}. In Theorem 5.2.1, we show that our two-phase strategy assures that no element in A_{k+1} has grown by more than the value of the largest diagonal element in A. This in turn means that our decomposition still achieves a good bound on $\|E\|_\infty$ in terms of the original matrix A.

THEOREM 5.2.1. *Let $A\in R^{n\times n}$, and let $\gamma=\max\{|A_{ii}|, 1\leqq i\leqq n\}$, $\xi=\max\{|A_{ij}|, 1\leqq i<j\leqq n\}$. Suppose we perform the standard Cholesky decomposition as described in* § 3 *for $k\geqq 1$ iterations, yielding the remaining principal submatrix $A_{k+1}\in R^{(n-k)\times(n-k)}$ (whose elements are denoted $(A_{k+1})_{ij}$, $k+1\leqq i,j\leqq n$), and let $\hat{\gamma}=\max\{|(A_{k+1})_{ii}|, k+1\leqq i\leqq n\}$ and $\hat{\xi}=\max\{|(A_{k+1})_{ij}|, k+1\leqq i<j\leqq n\}$. Then if $(A_{k+1})_{ii}\geqq 0$, $k+1\leqq i\leqq n$, then $\hat{\gamma}\leqq\gamma$ and $\hat{\xi}\leqq\xi+\gamma$.*

Proof. Let $A=\left[\begin{smallmatrix} B & C^T\\ C & F\end{smallmatrix}\right]$, where $B\in R^{k\times k}$, $C\in R^{(n-k)\times k}$, $F\in R^{(n-k)\times(n-k)}$. After k iterations of the Cholesky factorization, the first k columns of the Cholesky factor L have been determined; denote them by $\left[\begin{smallmatrix}\bar{L}\\ M\end{smallmatrix}\right]$ where $\bar{L}\in R^{k\times k}$ is triangular and $M\in R^{(n-k)\times k}$. Then

$$B=\bar{L}\bar{L}^T,\quad C=M\bar{L}^T,\quad F=MM^T+A_{k+1}. \tag{5.2.1}$$

From (5.2.1), $F_{ii}=\|M_{\text{row}i}\|_2^2+(A_{k+1})_{ii}$, $k+1\leqq i\leqq n$, so that from $F_{ii}\leqq\gamma$ and $(A_{k+1})_{ii}\geqq 0$,

$$\|M_{\text{row}i}\|_2^2\leqq\gamma. \tag{5.2.2}$$

Thus for any off-diagonal element of A_{k+1}, (5.2.1), (5.2.2), and the definition of ξ imply

$$|(A_{k+1})_{ij}|\leqq|F_{ij}-(M_{\text{row}i})(M_{\text{row}j})^T|\leqq\xi+\gamma, \tag{5.2.3}$$

which shows that $\hat{\xi}\leqq\xi+\gamma$. Also for all the diagonal elements of A_{k+1}, $(A_{k+1})_{ii}\geqq 0$, (5.2.1), and the definition of γ imply

$$0\leqq(A_{k+1})_{ii}\leqq F_{ii}\leqq\gamma, \tag{5.2.4}$$

which shows that $\hat{\gamma}\leqq\gamma$ and completes the proof. □

We note that the result of Theorem 5.2.1 is independent of the diagonal pivoting strategy that is used. We also note, however, that the technique of proof of Theorem 5.2.1 actually shows that the largest off-diagonal element in A_{k+1} is at most equal to the largest off-diagonal in F plus the largest diagonal in F, where F, as defined in the proof of Theorem 5.2.1, is the diagonal submatrix of A that corresponds to A_{k+1}. Thus a pivoting strategy that uses the larger diagonal elements as pivots in the first phase will limit the growth in the off-diagonal of A_{j+1} even more than is indicated by Theorem 5.2.1. Our phase-one algorithm pivots the largest remaining diagonal element to the top, and thus is likely to have this effect of further limiting element growth.

The possibility of incorporating this two-phase strategy into the method of GMW81 is discussed in the next section.

5.3. The complete new algorithm. We have now presented all the main parts of our new modified Cholesky decomposition algorithm. An outline of the complete algorithm is given in Algorithm 5.3.1, and a fully detailed description is given in Appendix 1. To summarize, the first phase of the algorithm applies the standard Cholesky decomposition, using a diagonal pivoting strategy that pivots the largest remaining diagonal element to the top left. This phase ends when the next iteration of the standard Cholesky decomposition would cause any diagonal element in the remaining submatrix to become nonpositive. In the second phase, the modified Cholesky decomposition described in § 5.1 is applied to the remaining submatrix. This phase determines what to add to the diagonal at each iteration from the lower Gerschgorin bound of the pivot row, and pivots based upon estimates of these lower Gerschgorin bounds.

Three additional, relatively minor features have been incorporated into Algorithm 5.3.1 to guard against the resultant $A+E$ being singular or very ill-conditioned:

ALGORITHM 5.3.1. Modified Cholesky Decomposition.

```
Given $A \in R^{n\times n}$ symmetric and $\tau$ (e.g., $\tau = (\text{macheps})^{1/3}$),
    find factorization $LL^T$ of $A+E$, $E \geqq 0$
$\gamma := \max_{1\leqq i\leqq n} |A_{ii}|$; $j := 1$
(*Phase one, $A$ potentially positive-definite*)
    While $j \leqq n$ do
        Pivot on maximum diagonal of remaining submatrix
        If $\min_{j+1\leqq i\leqq n} \{A_{ii} - A_{ij}^2/A_{jj}\} < \tau\gamma$
            then go to phase two
            else perform $j$th iteration of standard Cholesky factorization and
            increment $j$
(*Phase two, $A$ not positive-definite*)
    $k := j-1$ (*$k$ = number of iterations performed in phase one*)
    Calculate lower Gerschgorin bounds of $A_{k+1}$
    For $j := k+1$ to $n-2$ do
        Pivot on maximum lower Gerschgorin bound estimate
        Calculate $E_{jj}$ and add to $A_{jj}$
            $\left( {}^{*}E_{jj} = \max\left\{0, -A_{jj} + \max\left\{\sum_{i=j+1}^{n} |A_{ij}|, \tau\gamma\right\}, E_{j-1,j-1}\right\}^{*}\right)$
        update Gerschgorin bound estimates
        perform $j$th iteration of factorization
    complete factorization of final $2\times 2$ submatrix using its eigenvalues
```

First, the switch to phase two is made when any diagonal element of the remaining submatrix would become less than $\tau\gamma$, rather than less than zero as is discussed in § 5.2. Here γ is again the maximum diagonal of A, and τ is a small constant (we choose $\tau = (\text{macheps})^{1/3}$). This means we may perturb a positive-definite matrix if its condition number is greater than $1/\tau$. Second, in phase two, to assure that $A+E$ is positive-definite rather than positive semidefinite, we set (using the notation of § 5.1) each

$$\delta_j = \max\{0, -\alpha_j + \max\{\|a_j\|_1, \tau\gamma\}, \delta_{j-1}\},$$

where the $\tau\gamma$ term is new. This causes the bound (5.1.3) on $\|E\|_\infty$ to increase a tiny bit, to

$$\|E\|_\infty \leqq \text{Maxadd}_{\text{GCT}} + \tau\gamma, \tag{5.3.1}$$

but in conjunction with the preceding change, allows us to bound the condition number of $A+E$. Finally, at the final iteration of phase two, when only a 2×2 submatrix A_{n-1} remains, we use a different strategy: we calculate the eigenvalues λ_{lo} and λ_{hi} of A_{n-1}, and δ_{n-1} is chosen as the smallest nonnegative number so that $\delta_{n-1} \geqq \delta_{n-2}$, the l_2 condition number of $A_{n-1} + \delta_{n-1} I \leqq 1/\tau$, and $\lambda_{\text{lo}} + \delta_{n-1} \geqq \tau\gamma$. This generally gives a smaller value of δ_{n-1} than the Gerschgorin circle theorem based strategy would, and in theory it is straightforward to show that

$$\begin{aligned}\delta_{n-1} &= \max\left\{\delta_{n-2}, -\lambda_{\mathrm{lo}} + \max\left\{\frac{\tau(\lambda_{\mathrm{hi}}-\lambda_{\mathrm{lo}})}{1-\tau}, \tau\gamma\right\}\right\} \\ &\leqq \frac{1+\tau}{1-\tau}\,\mathrm{Maxadd}_{\mathrm{GCT}} + \frac{2\tau}{1-\tau}\,\gamma\end{aligned} \tag{5.3.2}$$

since $-\lambda_{\mathrm{lo}} \leqq \mathrm{Maxadd}_{\mathrm{GCT}}$ and $\lambda_{\mathrm{hi}} - \lambda_{\mathrm{lo}} \leqq 2\ (\mathrm{Maxadd}_{\mathrm{GCT}} + \gamma)$.

The theoretical properties of our full algorithm are summarized in Theorem 5.3.2.

THEOREM 5.3.2. *Let A, γ, and ξ be defined as in Theorem* 5.2.1, *suppose we apply the modified Cholesky factorization algorithm in Appendix* 1 *to A, resulting in the factorization LL^T of $A+E$. If A is positive-definite and at each iteration, $L_{jj}^2 \geqq \tau\gamma$, then $E=0$. Otherwise, E is a nonnegative diagonal matrix, with*

$$\|E\|_\infty \leqq \mathrm{Gersch} + \frac{2\tau}{1-\tau}(\mathrm{Gersch} + \gamma), \tag{5.3.3}$$

where Gersch *is the maximum of the negative of the lower Gerschgorin bounds of A_{k+1} that are calculated at the start of phase two. If $k=0$ then*

$$\mathrm{Gersch} = \mathrm{Maxadd}_{\mathrm{GCT}} \leqq \gamma + (n-1)\xi, \tag{5.3.4}$$

where $\mathrm{Maxadd}_{\mathrm{GCT}}$ *is given by* (5.1.1)-(5.1.2), *otherwise*

$$\mathrm{Gersch} \leqq [n-(k+1)](\gamma+\xi). \tag{5.3.5}$$

Proof. The proof is immediate from Theorem 5.1.2, Theorem 5.2.1, and equations (5.3.1)-(5.3.2). □

It is also possible to produce an upper bound on the condition number of $A+E$, of the same sort that is provable for the GMW81 algorithm. The key properties needed for this are that $\|E\|$, and hence $\max\{L_{ii}\}$, is bounded above, that $\min\{L_{ii}\}$ is bounded below (by $\sqrt{\tau\gamma}$), and that $|L_{ij}| < L_{ii}$ for all $1 \leqq j < i \leqq n$. (The final property comes from diagonal pivoting and the look-ahead property in phase one, and from the Gerschgorin bound strategy for choosing δ_j in phase two.) The bound on the condition number that we can obtain is of mainly theoretical interest, since it is exponential in n; the computational results of § 6 show that the condition number of $A+E$ is bounded above by about $1/\tau$ in practice.

We note that our two-phase strategy could also be incorporated into the method of GMW81, and that this would result in a significant improvement in their upper bound on $\|E\|_\infty$. This could be done by using the same two-phase structure, and replacing our phase two by their modified Cholesky decomposition. If this were done, their algorithm could simply choose $\beta^2 = \hat{\xi}/\sqrt{(n-k)^2-1}$ in phase two, rather than the maximum of this quantity and $\hat{\gamma}$ (where $\hat{\xi}$ and $\hat{\gamma}$ are defined as in Theorem 5.2.1) because it would know that it is dealing with a nonpositive-definite matrix. Hence the resultant method would achieve the bounds (4.3)-(4.4) if it switched to phase two immediately, and

$$\|E\|_\infty \leqq 4(n-k)\hat{\xi} + 2\hat{\gamma} \leqq 4(n-k)(\xi+\gamma) + 2\gamma$$

otherwise. This would be a significant improvement over the current bound (4.5), although it is still inferior to (5.3.3)-(5.3.5).

Our new algorithm meets our goal of not significantly increasing the cost of the standard Cholesky decomposition, which is about $n^3/6$ each additions and multiplications. The additional costs of the modified factorization are $(n-k)^2$ additions to calculate the Gerschgorin bounds of A_{k+1} at the start of phase two (where k is the number of iterations performed in phase one), $(n-k)^2/2$ additions to calculate the l_1 norms of the pivot rows during phase two, and at most $(n-k)^2/2$ each multiplications and additions to update the Gerschgorin bounds during phase two. In addition there is a small multiple of $n-k$ additional work. (The strategy for precalculating the new diagonal during phase one, in order when to determine when to switch to phase two, only costs a small multiple of n operations as long as the precalculated values are stored and used when phase one is continued.) Thus the total additional cost of the modified Cholesky decomposition is at most $2n^2$ additions and $n^2/2$ multiplications, in the case when phase two is started immediately ($k=0$). In many cases in our experience k is close to n so the additional costs are very small.

We have not performed a rounding error analysis of our modified Cholesky factorization. (To our knowledge, no such analysis has been performed for the method of GMW81 either.) It seems likely to us that the factorization should have similar finite precision properties to the standard Cholesky factorization (see, e.g., Wilkinson [5], [6]).

Finally, we include a small worked example to demonstrate the performance of the new modified Cholesky algorithm. Consider the matrix used by GMW81 to illustrate their modified Cholesky factorization:

$$A=\begin{bmatrix}1 & 1 & 2\\ 1 & 1 & 3\\ 2 & 3 & 1\end{bmatrix}.$$

Our new algorithm will proceed as follows. At the first iteration, no pivoting is performed in phase one, and then the algorithm immediately switches to phase two because $A_{33}-A_{31}^2/A_{11}<0$. The Gerschgorin intervals of A are

$$[-2,4],\quad [-3,5],\quad [-4,6].$$

The row with the maximum lower Gerschgorin bound is also row 1, so no pivoting is required in this iteration for phase two either. The modified Cholesky algorithm then choses $\delta=2=-$(Gerschgorin lower bound of row 1), and after the elimination step,

$$A_2=\begin{bmatrix}2/3 & 7/3\\ 7/3 & -1/3\end{bmatrix},$$

and the estimated Gerschgorin bounds are unchanged. The algorithm now enters the final, 2×2 submatrix stage. The eigenvalues of A_2 are $(-2.2196, 2.5538)$, so that $\delta_2=2.2196$ and $\delta total=2.2196$. Thus for the new algorithm,

$$E=\begin{bmatrix}2 & & \\ & 2.22 & \\ & & 2.22\end{bmatrix}$$

and $\|E\|_\infty=2.22$. This is one percent greater than the magnitude of the most negative eigenvalue of A, which is 2.2109. (If we had continued the Gerschgorin strategy for A_2 rather than use the eigenvalue strategy, δ_2 would be 2.67.)

Using the same matrix A, the GMW81 algorithm computes

$$E=\begin{bmatrix}2.77 & & \\ & 5.01 & \\ & & 2.24\end{bmatrix},$$

with $\|E\|_\infty=5.01$.

6. Computational results. We have compared the performance of our new modified Cholesky factorization (Algorithm 5.3.1 and Appendix 1) to the algorithm of GMW81 on a number of indefinite test matrices. The measures we used to assess the performance of the algorithms are the ratios $\|E\|_\infty/|\lambda_1(A)|$, termed "relative maxadd," which reflect how well the algorithm has satisfied the goal of adding as little as possible to the diagonal of A, and the condition numbers of $A+E$. We already know that the other two goals stated at the beginning of § 3, low cost and not disturbing safely positive-definite matrices, are satisfied by both algorithms.

We tested both algorithms on matrices of dimension 25, 50, and 75, with eigenvalue ranges of $[-1, 10000]$, $[-1, 1]$, and $[-10000, -1]$. For each combination of dimension and eigenvalue range, 10 matrices were created. Thus (the same) 90 test problems were used to test each algorithm. Each test matrix was created by forming the product $Q_1Q_2Q_3D\ (Q_1Q_2Q_3)^T$, where each Q_i is a Householder matrix of the form

$$Q_i = I - \left[\frac{2}{\|w\|_2^2}\, ww^T\right],$$

and each component of each w is randomly generated from a uniform distribution in the range $[-1, 1]$. Each D is a diagonal matrix whose elements were randomly generated from a uniform distribution in the desired eigenvalue range, with the exception that for the set of test matrices with eigenvalue range $[-1, 10000]$, one element of D was generated from the range $[-1, 0]$, thus guaranteeing at least one negative eigenvalue in the test matrices of that range.

The relative maxadds for the 90 tests of each algorithm are shown in Figs. 1(a), (c), (e), 2(a), (c), (e), and 3(a), (c), (e) in Appendix 2. In summary, the relative maxadds for the new algorithm were always small, and sometimes considerably superior to those for the GMW81 algorithm, although this algorithm's performance was also good in most cases. The relative maxadds for the new algorithm ranged from 1.06 to 2.5, and was below 1.71 for all but five of the 90 cases. The relative maxadds for the GMW81 algorithm ranged from 1.6 to 77.8, distributed as follows among the various groups of test matrices. For the matrices with eigenvalues in the $[-1, 10000]$ range, the relative maxadds ranged from 2.1 to 5.6. In the $[-1, 1]$ eigenvalue range, the relative maxadds were in the range 4.9 to 77.8, and in the final $[-10000, -1]$ eigenvalue range the relative maxadds ranged from 1.6 to 5.1. Comparing on a problem by problem basis, the new algorithm performed from 3.5 to 60.9 times better than the GMW81 method in terms of the relative maxadd for the problems with the $[-1, 1]$ eigenvalue range, and from 1.3 to 4.2 times better for the remaining test cases.

Figures 4(a)-4(i) show the relative maxadds for the new algorithm only, to illustrate more clearly how close $\|E\|_\infty$ is to $-\lambda_1(A)$ for this method. Also included in Figs. 4(a)-4(i) are the results for a version of the new algorithm that differs only in that it bases its pivots at each iteration of phase two upon the actual Gerschgorin bounds rather than their estimates. The additional cost of calculating these bounds is about $(n-k)^3/3$, or at most $n^3/3$, additional additions. The results in Fig. 4 show that pivoting on the exact Gerschgorin bounds leads to some improvement in the size of relative maxadd, but we do not consider the improvements sufficient to warrant the extra cost in general.

The condition numbers of $A+E$ for the two methods are given in Figs. 1(b), (d), (f), 2(b), (d), (f), and 3(b), (d), (f) in Appendix 2. Basically, both methods produced acceptably conditioned matrices in all cases. The condition numbers for the matrices produced by the new method varied from 10^1 to 10^6, whereas the condition numbers for the GMW81 method varied from 10^1 to 10^8. The condition numbers for the new method are sometimes directly related to the final step of the algorithm, which, if it increases $\|E\|_\infty$, does so by the amount necessary to make the final 2×2 submatrix positive-definite with condition number τ. In our test cases, the tolerance τ was $(\text{macheps})^{1/3}$, or roughly $10^{-5.2}$ on the Sun 3/75 used for these tests. This accounts for the condition numbers of almost 10^6 in all the cases where the final step increased $\|E\|_\infty$. Decreasing this tolerance generally was found to decrease the condition number, usually without appreciably increasing $\|E\|_\infty$.

Interestingly, in the cases where the new algorithm produced the most significant improvements in relative maxadds, the test problems with the $[-1, 1]$ eigenvalue range, it also produced much better conditioned matrices than the GMW81 algorithm. For this test set, the ratios of the GMW81 condition numbers to the condition numbers of the new algorithm were between 10^2 and 10^4 for $n=25$, between 10^4 and 10^5 for $n=50$, and between 10^5 and 10^7 for $n=75$. For the other two eigenvalue ranges, the ratios of the condition numbers produced by the two algorithms all varied by at most two orders of magnitude, with the condition numbers for the new algorithm consistently higher for the test problems in the $[-1, 10000]$ eigenvalue range, and the GMW81 condition numbers usually higher for the test problems in the $[-10000, -1]$ range.

Finally, Figs. 5(a), (b) in Appendix 2 contain the test results for a different set of matrices of dimension $n=25$ with eigenvalue range $[-1, 10000]$. The difference between these test matrices and the ones used in Figs. 1(a), (b) is that these matrices were created to have at least three negative eigenvalues, whereas the original test problems in the $[-1, 10000]$ range were created with at least 1 negative eigenvalue. What is interesting about the results of this new test set is that on one particular matrix out of the 10, the new algorithm performs significantly *worse* than the GMW81 algorithm. (This phenomenon did *not* occur with the test sets of size 50 or 75 in this range with three negative eigenvalues, so we have not included this data.) The poor behavior occurred when the algorithm was at the $(n-4)$th iteration, so we created a 4×4 matrix with similar characteristics that illustrates the problem even more markedly.

The matrix

$$A=\begin{bmatrix} 1{,}890.3 & -1{,}705.6 & -315.8 & 3{,}000.3 \\ -1{,}705.6 & 1{,}538.3 & 284.9 & -2{,}706.6 \\ -315.8 & 284.9 & 52.5 & -501.2 \\ 3{,}000.3 & -2{,}706.6 & -501.2 & 4{,}760.8 \end{bmatrix}$$

has eigenvalues -0.378, -0.343, -0.248, and 8242.869. The first few steps performed by the new algorithm are as follows:

(1) Interchange row and column 4 with row and column 1, because $A_{4,4}$ is the maximum diagonal element.

(2) Switch to phase two because $A_{3,3}-(A_{3,1})^2/A_{1,1}<0$.

(3) Calculate the lower Gerschgorin bounds $\{-1447.3, -3158.8, -1049.4, -3131.4\}$, and since $-G\text{low}_3$ is the maximum value, interchange row and column 3 with row and column 1.

(4) Add $(-G\text{low}_{\text{pivotrow}})=1049.4$ to $A_{1,1}$.

At this point in the computation, the new algorithm has already added much more to the diagonal than is necessary to make A positive-definite. From this point on it does not increase $\|E\|_\infty$, so that the final value of $\|E\|_\infty$ is 1049.4. On the other hand, the GMW81 algorithm produces $\|E\|_\infty=1.03$. This behavior occurs because, at the first iteration, the GMW81 algorithm pivots on the maximum diagonal element and then adds nothing to the diagonal, which after elimination results in a 3×3 submatrix all of whose entries have absolute value less than 0.52. This is guaranteed to then lead to a small $\|E\|_\infty$. (Indeed, if our algorithm performed the same first step as the GMW81 algorithm and then proceeded as usual, it would produce $\|E\|_\infty = 0.665$.)

The essential characteristic of this example is that A is equal to a large symmetric rank-one matrix plus a small indefinite matrix. Thus, if nothing is added to A_{11} at the first iteration, the remaining submatrix after the elimination has very small elements, and $\|E\|$ is small. The GMW81 algorithm will usually outperform ours on matrices of this type. We have experimented with modifications to our algorithm that perform well for this case, but all of them resulted in degradation of our algorithm's performance in other cases. Since the case only occurred once in the 120 test cases discussed in this section, we would hope that it is not common in practice.

7. Summary and conclusions. We have presented a new modified Cholesky factorization algorithm that does a good job of meeting the objectives outlined at the start of § 3. It is based upon two new techniques, the use of Gerschgorin circle theorem bounds to decide how much to add to the diagonal, and the use of a two-phase structure to differentiate between positive-definite and nonpositive-definite matrices. It costs at most $2n^2$ additions and $n^2/2$ multiplications more than the standard Cholesky factorization, and its theoretical bound on $\|E\|_\infty$ is a factor of n lower than for the GMW81 method. In computational tests on nonpositive-definite matrices, it virtually always produces a smaller $\|E\|_\infty$ than the method of GMW81, and the conditioning of $A+E$ is always quite acceptable. On the class of test problems where the GMW81 algorithm had the most difficulty, those with eigenvalue range $[-1, 1]$, the decreases in $\|E\|_\infty$ and in the condition number of $A+E$ are both substantial.

In our computational tests, both our method and that of GMW81 virtually always produce values of $\|E\|_\infty$ that are orders of magnitude smaller than the worst-case theoretical bounds. Empirically, this seems to occur because the matrix elements, and hence $\|E\|_\infty$ do not grow nearly as quickly as in the worst-case analysis. This disparity between theory and practice makes it unclear whether the practical improvement of our method over the method of GMW81 is tied to its theoretical improvement. We believe that it is, for two reasons. First, basing the amount to add on the l_1 norm of the pivot row rather than the l_∞ norm may cause us to add less, and second, separating the two phases of the algorithm may allow us to add less in practice as well as save a factor of n in theory. A more rigorous explanation would be useful.

We have not tested the effect of substituting our new modified Cholesky factorization for that of GMW81 in optimization algorithms. The most common optimization test problems have small n and few if any indefinite iterations, so probably there would be little effect on these. The new algorithm might make a difference on problems where n is larger and there is some indefiniteness. In our opinion, the biggest advantage of the new method for optimization purposes is its improved theoretical bound on $\|E\|_\infty$ and the corresponding reduction in $\|E\|_\infty$ that has been observed in practice. These properties guard against overflows during the factorization, and against steps $(A+E)^{-1}\nabla f(x)$ that are far too small.

In addition, the new algorithm leads to an easy implementation of trust region methods for optimization, because $\|E\|_\infty$ is generally within a factor of 1.5 of the negative of the smallest eigenvalue $\lambda_1(A)$ of A. By first calculating E, then replacing A with $A+(\|E\|_\infty)I$ if $E \neq 0$, and then using the trust region method for positive definite matrices, we will usually get the solution to the exact, possibly indefinite trust region problem without using any other special provisions for dealing with nonpositive-definite matrices. We have already used the factorization successfully in this context. If there are other computational algorithms where a crude estimate of the most negative eigenvalue of a matrix is useful, either by itself or as a starting estimate of some iterative procedure, then this factorization may provide a good way to find it.

Finally, Dr. N. Gould of Harwell Laboratory, England, reports that our modified Cholesky factorization has proven useful to him for a different reason than those discussed above. He is using it in a large, sparse optimization code, where the linear system is solved by a multifrontal method, and diagonal pivoting during the modified Cholesky factorization is unnecessary due to the properties of the Hessian matrices. In this case, our method has the advantage that it does not require the full matrix to be known a priori, so that it may be assembled incrementally, with only the front and the diagonals needed in storage at any given time. In contrast, in the GMW81 method, the entire matrix must be known during the initialization phase to calculate the terms γ and ξ in the notation of § 3. Gould has implemented an unpivoted version of our factorization in this code and reports very satisfactory performance.

Appendix 1. Complete Modified Cholesky Decomposition Algorithm.
Given $A \in R^{n\times n}$ symmetric (stored in lower triangle) and τ (e.g., $\tau = (\text{macheps})^{1/3}$), find factorization LL^T of $A+E$, $E \geqq 0$

```
phaseone := true
$\gamma := \max_{1\leqq i\leqq n} |A_{ii}|$
$j := 1$
(*Phase one, $A$ potentially positive-definite*)
While $j \leqq n$ and phaseone = true do
    (*Pivot on maximum diagonal of remaining submatrix*)
        $i :=$ index of $\max_{j\leqq i\leqq n} A_{ii}$
        if $i \neq j$, switch rows and columns $i$ and $j$ of $A$
    if $\min_{j+1\leqq i\leqq n} \{A_{ii} - A_{ij}^2/A_{jj}\} < \tau\gamma$
        then phaseone := false (*go to phase two*)
        else (*perform $j$th iteration of factorization*)
            $L_{jj} = \sqrt{A_{jj}}$ (*$L_{jj}$ overwrites $A_{jj}$*)
            For $i := j+1$ to $n$ do
                $L_{ij} := A_{ij}/L_{jj}$ (*$L_{ij}$ overwrites $A_{ij}$*)
                For $k := j+1$ to $i$ do
                    $A_{ik} := A_{ik} - L_{ij}L_{kj}$
            $j := j+1$
(*end phase one*)

(*Phase two, $A$ not positive-definite*)
If phaseone = false then
    $k := j-1$ (*$k$ = number of iterations performed in phase one*)
    (*Calculate lower Gerschgorin bounds of $A_{k+1}$*)
        For $i := k+1$ to $n$ do
            $g_i := A_{ii} - \sum_{j=k+1}^{i-1} |A_{ij}| - \sum_{j=i+1}^{n} |A_{ji}|$

    (*Modified Cholesky Decomposition*)
    For $j := k+1$ to $n-2$ do
        (*Pivot on maximum lower Gerschgorin bound estimate*)
            $i :=$ index of $\max_{j\leqq i\leqq n} \{g_i\}$
            if $i \neq j$, switch rows and columns $i$ and $j$ of $A$
        (*Calculate $E_{jj}$ and add to diagonal*)
            $normj := \sum_{i=j+1}^{n} |A_{ij}|$
            $\delta (* = E^*_{jj}) = \max\{0, -A_{jj} + \max\{normj, \tau\gamma\}, \delta prev\}$
            if $\delta > 0$ then
                $A_{jj} := A_{jj} + \delta$
                $\delta prev := \delta$ (*$\delta prev$ will contain $\|E\|^*_\infty$)
        (*update Gerschgorin bound estimates*)
        If $A_{jj} \neq normj$ then
            $temp := 1 - normj / A_{jj}$
            for $i := j+1$ to $n$ do
                $g_i := g_i + |A_{ij}|^*\ temp$
        (*perform $j$th iteration of factorization*)
            same code as in phase one
    (*final $2 \times 2$ submatrix*)
        $\lambda_{\text{lo}}, \lambda_{\text{hi}} :=$ eigenvalues of $\begin{bmatrix} A_{n-1,n-1} & A_{n,n-1} \\ A_{n,n-1} & A_{n,n} \end{bmatrix}$
        $\delta := \max\{0, -\lambda_{\text{lo}} + \tau^* \max\{1/(1-\tau)(\lambda_{\text{hi}} - \lambda_{\text{lo}}), \gamma\}, \delta prev\}$
        if $\delta > 0$ then
            $A_{n-1,n-1} := A_{n-1,n-1} + \delta$
            $A_{n,n} := A_{n,n} + \delta$
            $\delta prev := \delta$
        $L_{n-1,n-1} := \sqrt{A_{n-1}, A_{n-1}}$ (*overwrites $A_{n-1,n-1}$*)
        $L_{n,n-1} := A_{n,n-1} / L_{n-1,n-1}$ (*overwrites $A_{n,n-1}$*)
        $L_{n,n} := (A_{n,n} - L^2_{n,n-1})^{1/2}$(*overwrites $A_{n,n}$*)
(*End phase two*)
```

Appendix 2. Computational results are given in Figs. 1-5.

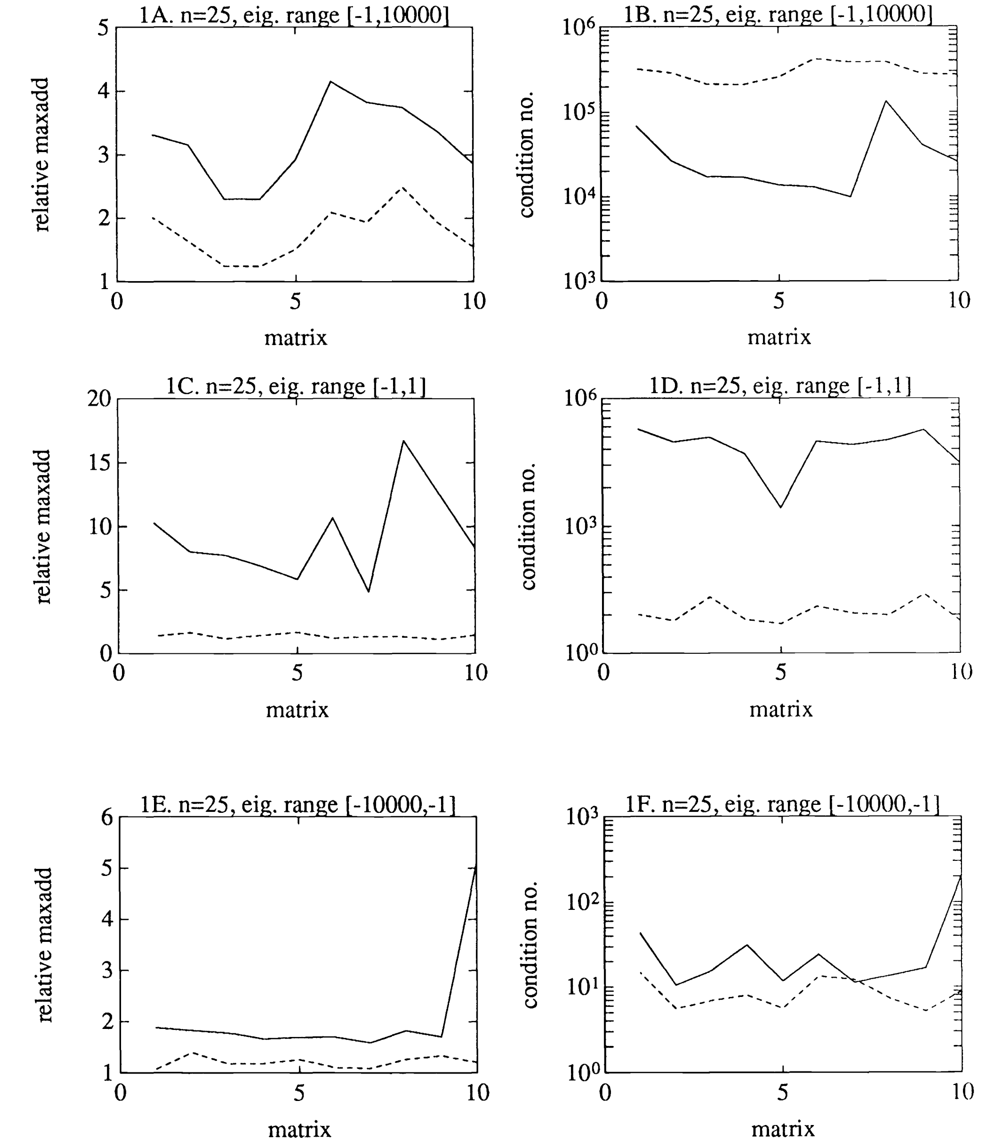


FIG. 1. *Performance of existing and new methods on* 10 *indefinite matrices with* $n = 25$. *Methods*: *Gill, Murray, and Wright* ——; *new method* - - - -. *Note*: *Relative* maxadd = (*maximum added to diagonal*)/(*-smallest eigenvalue*).

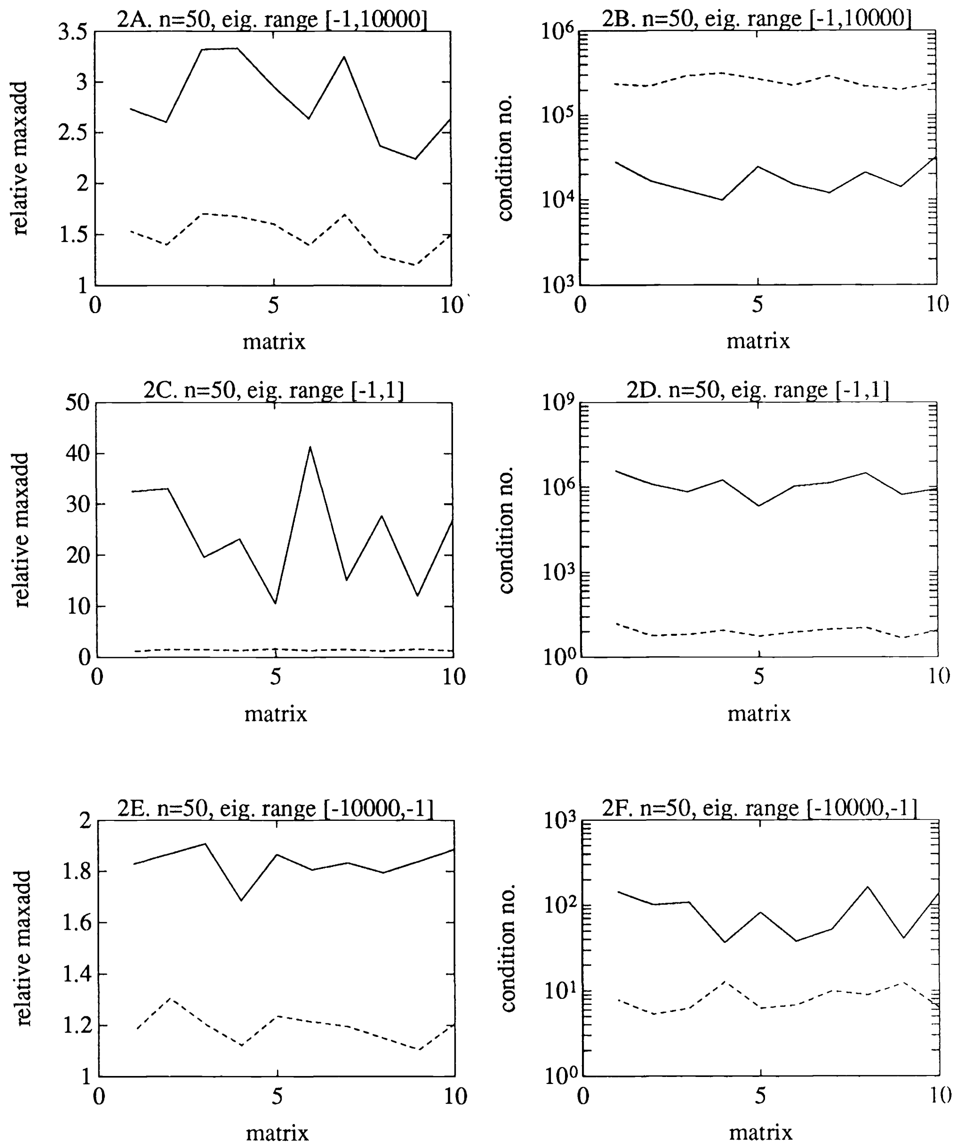


FIG. 2. *Performance of existing and new methods on* 10 *indefinite matrices with* $n = 50$. *Methods*: *Gill, Murray, and Wright* ——; *new method* - - - -, *Note*: *Relative* maxadd = (*maximum added to diagonal*)/(*-smallest eigenvalue*).

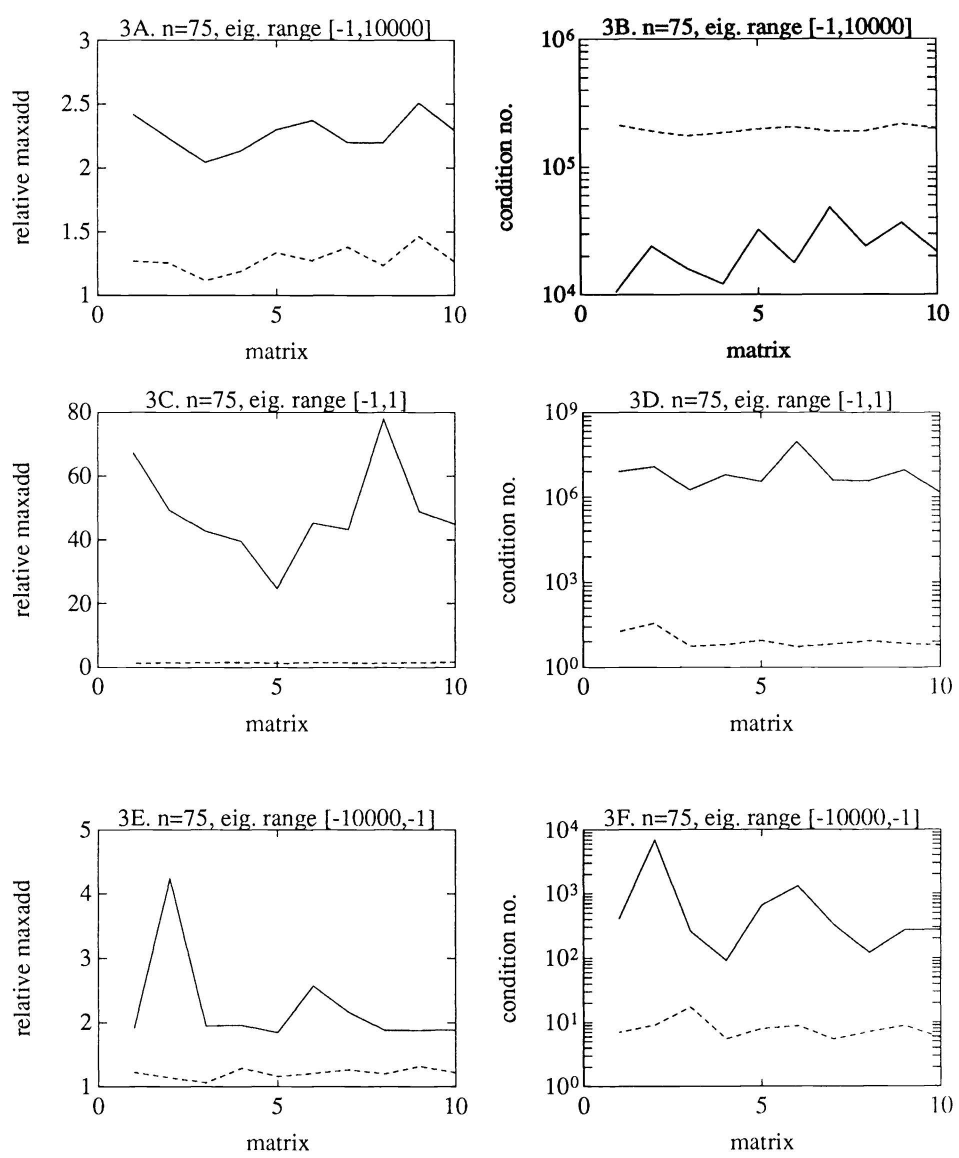


FIG. 3. *Performance of existing and new methods on* 10 *indefinite matrices with* $n = 75$. *Methods*: *Gill, Murray, and Wright* ——; *new method* - - - -. *Note*: *Relative* maxadd = (*maximum added to diagonal*)/(-*smallest eigenvalue*).

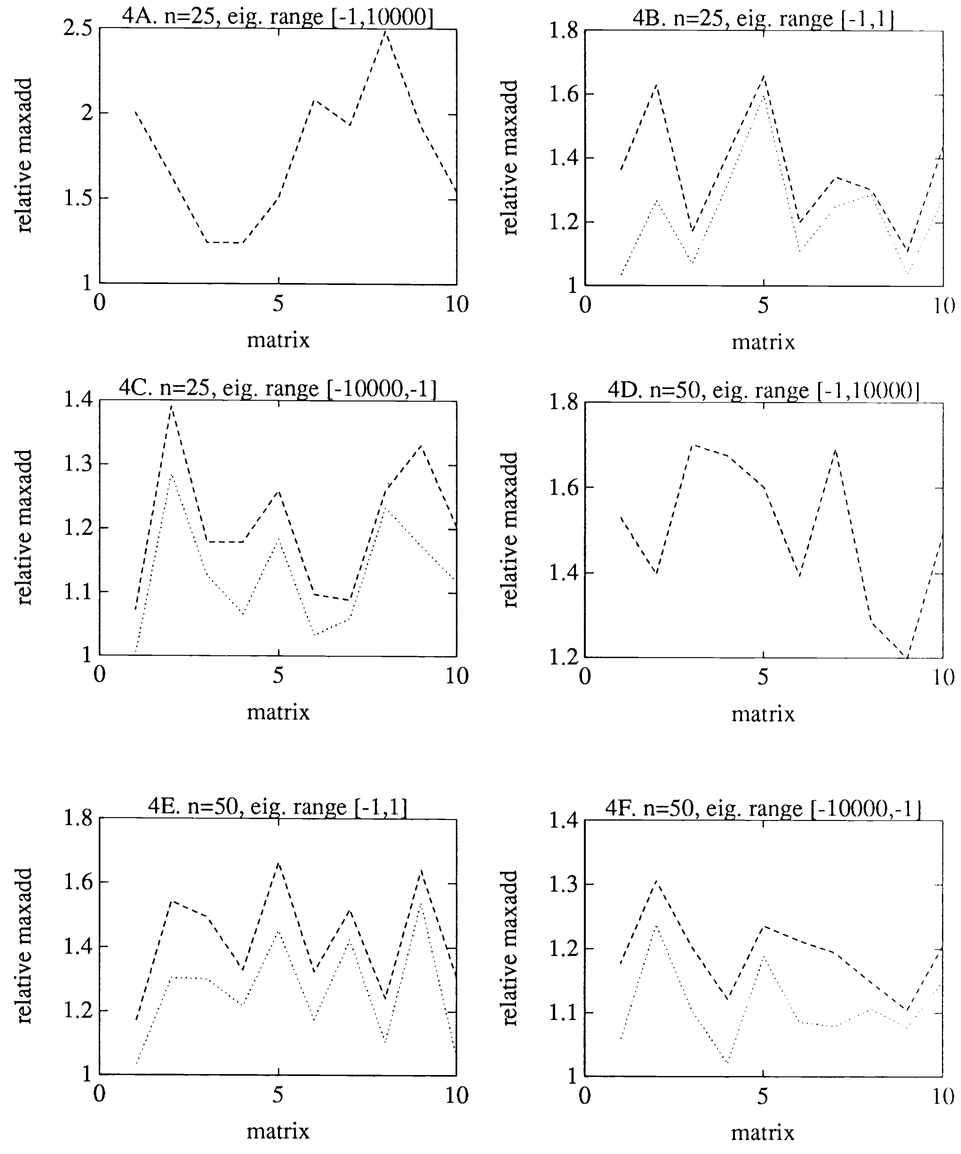


FIG. 4. *Relative* maxadds *for two versions of the new method. Methods: new method* - - - -; *new method with* $O(n^{**}3)$ *pivoting* · · · · ·. (*Note. Two curves are identical in Figs.* 4A, D, *and* G.)

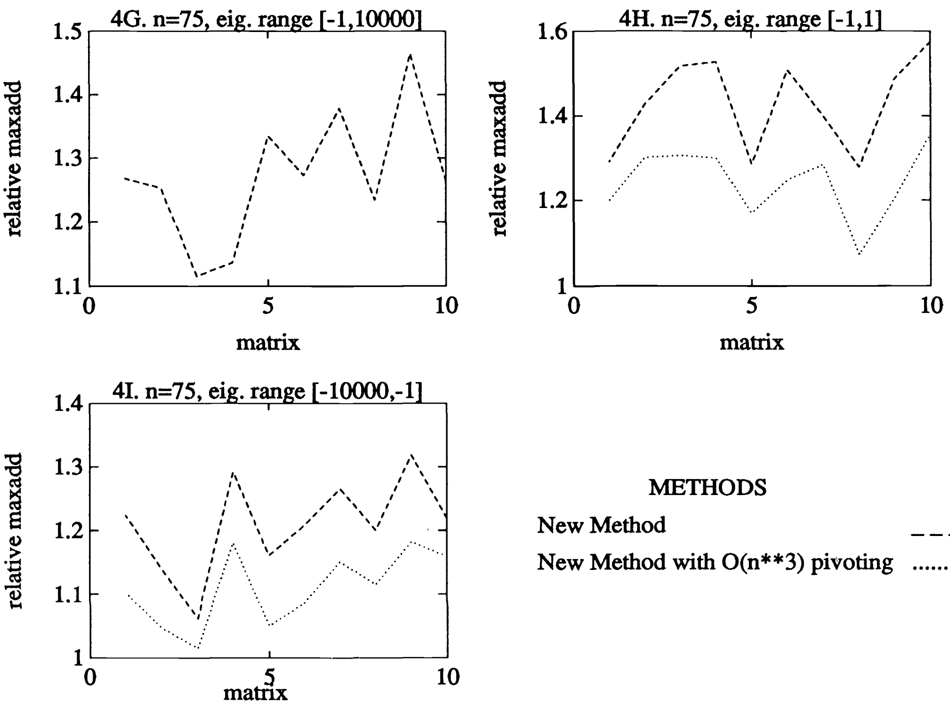


FIG. 4. *Continued.*

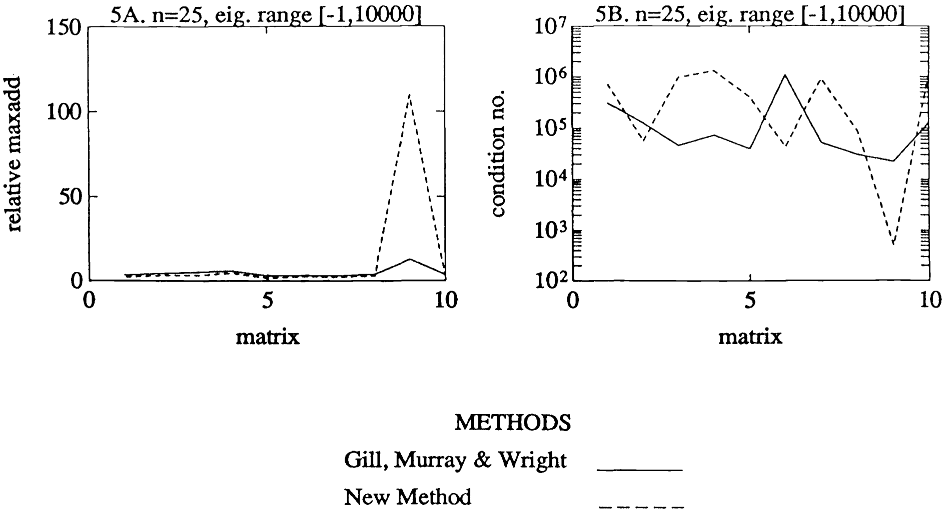


FIG. 5. *Performance of existing and new methods on a test set with three negative eigenvalues. Methods: Gill, Murray, and Wright* ——; *new method* - - - -.

Acknowledgments. We thank Eric Van Vleck for performing the early computational experiments with the new method, the referees for their careful reading of the paper, and Nick Gould for relating the information mentioned above.

REFERENCES

[1] J. E. DENNIS, JR. AND R. B. SCHNABEL, *Numerical Methods for Nonlinear Equations and Unconstrained Optimization*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

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

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

[4] G. A. SHULTZ, 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.

[5] J. H. WILKINSON, *Error analysis of direct methods of matrix inversion*, J. Assoc. Comput. Math., 8 (1961), pp. 281-330.

[6] ———, *Rounding Errors in Algebraic Processes*, Prentice-Hall, Englewood Cliffs, NJ, 1963.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 6, pp. 1159-1163, November 1990

PERTURBATION ANALYSIS OF MATRIX QUADRATIC EQUATIONS*

M. M. KONSTANTINOV†, P. HR. PETKOV‡, AND N. D. CHRISTOV‡

Abstract. Nonlocal perturbation analysis of a general type matrix quadratic equation is presented using the technique of contractive operators. The continuous-time algebraic matrix Riccati equation arising in linear-quadratic optimization is considered as a particular case.

Key words. perturbation analysis, matrix quadratic equations, algebraic Riccati equations

AMS(MOS) subject classifications. primary 15A24; secondary 93B35

In this paper we study the sensitivity of the regular solutions of a general matrix quadratic equation relative to perturbations in its coefficients. Upper bounds for norms of the perturbed solutions are obtained without the assumption that the coefficient perturbations are asymptotically small. The well-known continuous-time algebraic Riccati equation is considered as an example.

Consider the matrix quadratic equation

$$R(X)=A+\sum_{i=1}^{N}(B_iXC_i+D_iXE_iXF_i)=0 \tag{1}$$

where $A\in\mathbb{R}^{p\cdot q}$, B_i, $D_i\in\mathbb{R}^{p\cdot n}$, C_i, $F_i\in\mathbb{R}^{m\cdot q}$, $E_i\in\mathbb{R}^{m\cdot n}$, and $X\in\mathbb{R}^{n\cdot m}$ is the unknown matrix. In what follows we assume summation from $i=1$ to $i=N$ for the repeated subscript i, rewriting (1) as $A+B_iXC_i+D_iXE_iXF_i=0$. It is also supposed that there is at least one triple (D_i, E_i, F_i) with $D_i\neq 0$, $E_i\neq 0$, $F_i\neq 0$ since in the opposite case, (1) reduces to a linear equation.

To ensure that (1) is consistent and has isolated solutions, it is supposed that $mn=pq=k$.

Denote by $S\subset\mathbb{R}^{n\cdot m}$ the set of real solutions of (1). For $X\in S$ the tangent set (relative to S) at X is $X+\operatorname{Ker} T_X$, where $T_X:\mathbb{R}^{n\cdot m}\to\mathbb{R}^{p\cdot q}$ is defined from

$$T_X(Y)=B_iYC_i+D_i(XE_iY+YE_iX)F_i. \tag{2}$$

The solution $X\in S$ is said to be regular if $\operatorname{Ker} T_X=0$. The latter is valid if and only if

$$\det M_X\neq 0,\ M_X=C_i^T\otimes B_i+F_i^T\otimes(D_iXE_i)+(E_iXF_i)^T\otimes D_i\in\mathbb{R}^{k\cdot k}. \tag{3}$$

Let

$$t(X)=\min(\|T_X(Y)\|:\ \|Y\|=1). \tag{4}$$

Then $X\in S$ is regular if and only if $t(X)>0$.

Note that the function $t:\mathbb{R}^{n\cdot m}\to\mathbb{R}_+$ is a generalization of the function separating two square matrices [1]. Following similar arguments as in [1] it may be shown that if $\|\cdot\|$ is the Frobenius norm then $t(X)$ is the smallest singular value of M_X (equal to $\|M_X^{-1}\|_2$ if (3) holds).

* Received by the editors December 18, 1985; accepted for publication (in revised form) October 23, 1989.

† Institute of Engineering Cybernetics and Robotics, Akad. G. Bonchev Str., Bl. 2, 1113 Sofia, Bulgaria. Present address, Department of Mathematics, High School of Architecture and Civil Engineering (VIAS), 1 Hr. Smirnenski Blv., 1421 Sofia, Bulgaria.

‡ Department of Automatics, Higher Institute of Mechanical and Electrical Engineering, Bl. 2, 1156 Sofia, Bulgaria.

Now let $A^*, B_i^*, \cdots, F_i^*$ be perturbations of the corresponding matrix coefficients $A, B_i, \cdots, F_i$ in (1). Denote $a^* = \|A^*\|$, $b_i = \|B_i\|$, $b_i^* = \|B_i^*\|, \cdots, f_i = \|F_i\|, f_i^* = \|F_i^*\|$, and

$$\mu = (a^*, b_1^*, c_1^*, \cdots, b_N^*, c_N^*, d_1^*, e_1^*, f_1^*, \cdots, d_N^*, e_N^*, f_N^*)^T \in \mathbb{R}^{5N+1}.$$

Let $X \in S$ be a fixed regular solution of (1). The main problems to be solved in this paper are:

(i) Find conditions for μ such that the perturbed equation

$$(5) \quad U(Y) = A + A^* + (B_i + B_i^*)Y(C_i + C_i^*) + (D_i + D_i^*)Y(E_i + E_i^*)Y(F_i + F_i^*) = 0$$

has a solution $Y = X + X^*$, where the elements of X^* are analytic functions of the elements of the perturbation matrices $A^*, B_i^*, \cdots, F_i^*$, and $X^* = 0$ for $\mu = 0$.

(ii) Find an estimate of the form $\|X^*\| \leqq r(\mu)$, where $r: \mathbb{R}^{5N+1} \to \mathbb{R}_+$ is an analytic function, nondecreasing in each component of μ, and such that $r(0) = 0$.

Since

$$U(X + X^*) = R(X) + T_X(X^*) + V_X^*(X^*),$$

where

$$(6) \quad \begin{aligned} V_X^*(Z) &= A^* + K^*(X) + K^*(Z) + L^*(X, X) + L^*(X, Z) + L^*(Z, X) \\ &\quad + L^*(Z, Z) + L(Z, Z), \\ K^*(Z) &= B_i^* Z C_i + B_i Z C_i^* + B_i^* Z C_i^*, \\ L^*(Z_1, Z_2) &= D_i^* Z_1 E_i Z_2 F_i + D_i Z_1 E_i^* Z_2 F_i + D_i Z_1 E_i Z_2 F_i^* + D_i^* Z_1 E_i^* Z_2 F_i \\ &\quad + D_i^* Z_1 E_i Z_2 F_i^* + D_i Z_1 E_i^* Z_2 F_i^* + D_i^* Z_1 E_i^* Z_2 F_i^*, \\ L(Z_1, Z_2) &= D_i Z_1 E_i Z_2 F_i, \end{aligned}$$

and $R(X) = 0$, then for $Y = X + X^*$ equation (5) is reduced to

$$(7) \qquad T_X(X^*) + V_X^*(X^*) = 0.$$

In (6) $K^*: \mathbb{R}^{n \cdot m} \to R^{p \cdot q}$ is a linear operator, whereas $L^*, L: \mathbb{R}^{n \cdot m} \times \mathbb{R}^{n \cdot m} \to \mathbb{R}^{p \cdot q}$ are bilinear operators, such that

$$(8) \quad \begin{aligned} \|K^*(Z)\| &\leqq k^* \|Z\|, \\ \|L^*(Z_1, Z_2)\| &\leqq \ell^* \|Z_1\| \, \|Z_2\|, \\ \|L(Z_1, Z_2)\| &\leqq \ell \|Z_1\| \, \|Z_2\|, \end{aligned}$$

where

$$\begin{aligned} k^* &= (b_i + b_i^*)(c_i + c_i^*) - b_i c_i, \\ \ell^* &= (d_i + d_i^*)(e_i + e_i^*)(f_i + f_i^*) - d_i e_i f_i, \\ \ell &= d_i e_i f_i. \end{aligned}$$

When $\|\mu\|$ is small the constants k^*, ℓ^* are also small and of asymptotic order $O(\|\mu\|)$.

Rewrite (7) as an operator equation

$$(9) \qquad X^* = P_X^*(X^*)$$

where

$$P_X^* = -T_X^{-1} \circ V_X^*: \mathbb{R}^{n \cdot m} \to \mathbb{R}^{n \cdot m}.$$

We will show that under some conditions on μ there exists $r = r(\mu)$ such that P_X^* is contractive and maps the set $S_r = (Z: \|Z\| \leqq r)$ into itself.

Let $x = \|X\|$. Having in mind (6) and (8) for $Z \in S_r$, we have

$$\begin{aligned} \|V_X^*(Z)\| &\leqq \|A^*\| + \|K^*(X)\| + \|K^*(Z)\| + \|L^*(X,X)\| + \|L^*(X,Z)\| \\ &\quad + \|L^*(Z,X)\| + \|L^*(Z,Z)\| + \|L(Z,Z)\| \\ &\leqq a^* + k^*(x+r) + \ell^*(x+r)^2 + \ell r^2. \end{aligned} \tag{10}$$

Similarly, for $Z_1, Z_2 \in S_r$ it follows from (8) that

$$\begin{aligned} &\|K^*(Z_1) - K^*(Z_2)\| \leqq k^*\|Z_1 - Z_2\|, \\ &\|L^*(Z_1, X) - L^*(Z_2, X)\| \leqq \ell^* x\|Z_1 - Z_2\|, \\ &\|L^*(X, Z_1) - L^*(X, Z_2)\| \leqq \ell^* x\|Z_1 - Z_2\|, \\ &\|L^*(Z_1, Z_1) - L^*(Z_2, Z_2)\| = \|L^*(Z_1, Z_2) - L^*(Z_1, Z_2) + L^*(Z_1, Z_2) - L^*(Z_2, Z_2)\| \\ &\qquad \leqq \|L^*(Z_1, Z_1) - L^*(Z_1, Z_2)\| \\ &\qquad + \|L^*(Z_1, Z_2) - L^*(Z_2, Z_2)\| \\ &\qquad \leqq \ell^*(\|Z_1\| + \|Z_2\|)\|Z_1 - Z_2\| \leqq 2\ell^* r\|Z_1 - Z_2\|, \\ &\|L(Z_1, Z_1) - L(Z_2, Z_2)\| \leqq 2\ell r\|Z_1 - Z_2\|. \end{aligned}$$

Hence

$$\begin{aligned} \|V_X^*(Z_1) - V_X^*(Z_2)\| &\leqq \|K^*(Z_1) - K^*(Z_2)\| + \|L^*(X, Z_1) - L^*(X, Z_2)\| \\ &\quad + \|L^*(Z_1, X) - L^*(Z_2, X)\| \\ &\quad + \|L^*(Z_1, Z_1) - L^*(Z_2, Z_2)\| \\ &\quad + \|L(Z_1, Z_1) - L(Z_2, Z_2)\| \\ &\leqq (k^* + 2\ell^*(x+r) + 2\ell r)\|Z_1 - Z_2\|. \end{aligned} \tag{11}$$

In accordance with (4) we have $t(X)\|Y\| \leqq \|T_X(Y)\|$ for all $Y \in \mathbb{R}^{n \cdot m}$. Therefore

$$\|T_X^{-1}\| \leqq \frac{1}{t}, \qquad t = t(X). \tag{12}$$

Now the inequalities (10)-(12) yield

$$\begin{aligned} \|P_X^*(Z)\| &\leqq \|T_X^{-1}\|\,\|V_X^*(Z)\| \\ &\leqq (a^* + k^*(x+r) + \ell^*(x+r)^2 + \ell r^2)/t = g_0(r), \end{aligned} \tag{13}$$

$$\begin{aligned} \|P_X^*(Z_1) - P_X^*(Z_2)\| &\leqq \|T_X^{-1}\|\,\|V_X^*(Z_1) - V_X^*(Z_2)\| \\ &\leqq \frac{1}{t}(k^* + 2\ell^*(x+r) + 2\ell r)\|Z_1 - Z_2\| \\ &= g_1(r)\|Z_1 - Z_2\| \end{aligned} \tag{14}$$

for all $Z, Z_1, Z_2 \in S_r$. In view of (14), (13) the operator P_X^* is a contraction and maps the compact set S_r into itself if there exist $r > 0$ such that $g_0(r) \leqq r$ and $g_1(r) < 1$. The necessary and sufficient condition for the last three inequalities to hold is

$$k^* + 2\ell^* x + 2((\ell + \ell^*)(a^* + k^* x + \ell^* x^2))^{1/2} < t. \tag{15}$$

In this case we may choose

$$r = r(\mu) = (t - k^* - 2\ell^* x - ((t - k^* - 2\ell^* x)^2 - 4(\ell + \ell^*)(a^* + k^* x + \ell^* x^2))^{1/2})/(2(\ell + \ell^*)). \tag{16}$$

The above considerations are summarized in the following theorem.

THEOREM. *Let $X \in S$ and the conditions* (3) *and* (15) *be fulfilled. Then the perturbed equation* (5) *has a unique solution $y = X + X^*$ such that the elements of X^* are analytic functions of the elements of A^*, $B_i^*, \cdots, F_i^*$, and*

$$\|X^*\| \leqq r(\mu) \tag{17}$$

(*the uniqueness of $Y = X + X^*$ is claimed only in a $r(\mu)$-neighborhood of X*).

Note that the estimate (17), (16) is true for all μ satisfying (15), i.e., the norm $\|\mu\|$ of the perturbation vector μ need not be asymptotically small. If, however, $\|\mu\|$ is small, then it follows from (17), (16) that

$$\|X^*\| \leqq \frac{a^* + k_1^* x + \ell_1^* x^2}{t} + O(\|\mu\|^2), \qquad \|\mu\| \to 0,$$

where

$$k_1^* = b_i c_i^* + b_i^* c_i = O(\|\mu\|),$$

$$\ell_1^* = d_i e_i f_i^* + d_i e_i^* f_i + d_i^* e_i f_i = O(\|\mu\|).$$

If $X \neq 0$ (i.e., if $A \neq 0$) then the relative perturbation in the solution may be estimated as

$$\frac{\|X^*\|}{\|X\|} \leqq \frac{1}{t}\left(\frac{a^*}{\|X\|} + k_1^* + \ell_1^* \|X\|\right) + O(\|\mu\|^2).$$

An important application of the above perturbation analysis is the continuous-time algebraic Riccati equation

$$Q + G^T X + XG - XHX = 0 \tag{18}$$

arising in the theory of linear-quadratic optimization. Here $Q, G, H, X \in \mathbb{R}^{n,n}$; $Q = Q^T \geqq 0$; $H = H^T \geqq 0$ $(H \neq 0)$. If the pair $[G, H)$ is stabilizable and the pair $(Q, G]$ is detectable, then (18) has a unique solution $X = X^T \geqq 0$ (note that this is also the unique solution of (18) for which the matrix $G - HX$ is stable) [2].

Let Q^*, G^*, H^* be perturbations of Q, G, H, and $q^* = \|Q^*\|$, $g^* = \|G^*\|$, $h^* = \|H^*\|$, $h = \|H\|$. Then in the notation of the theorem,

$$k^* = 2g^*, \quad \ell^* = h^*, \quad \ell = h, \quad x = \|X\|, \quad \mu = (q^*, g^*, g^*, h^*)^T,$$

$$t = \min\,(\|(G - HX)^T Y + Y(G - HX)\| : \|Y\| = 1) > 0.$$

Hence we may formulate the following corollary.

COROLLARY. *Let the inequality*

$$2(g^* + h^* x + ((h + h^*)(q^* + 2g^* x + h^* x^2))^{1/2}) < t$$

be satisfied. Then the perturbed Riccati equation

$$Q + Q^* + (G + G^*)^T Y + Y(G + G^*) - Y(H + H^*) Y = 0$$

has a unique solution $Y = X + X^*$ such that the elements of X^* are analytic functions of the elements of Q^*, G^*, H^*, and

$$\|X^*\| \leqq r(\mu) = (t - 2g^* - 2h^*x - ((t - 2g^* - 2h^*x)^2 - 4(h + h^*) \times (q^* + 2g^*x + h^*x^2))^{1/2})/(2(h + h^*)) \tag{19}$$

(*the uniqueness of $X + X^*$ is claimed in the $r(\mu)$-neighborhood of X*).

It is worth mentioning that for arbitrary small $\|\mu\| > 0$ the solution $X + X^*$ may be nonsymmetric, and the matrix $X + (X^* + X^{*T})/2$ may not be positive semidefinite (the latter is possible only if the pair $(Q, G]$ is not observable).

For small $\|\mu\|$ the function r from (19) may be represented as

$$r(\mu) = \sum_{j=1}^{k} r_j(\mu) + O(\|\mu\|^{k+1}), \qquad r_j(\mu) = O(\|\mu\|^j). \tag{20}$$

The first three terms in the asymptotic series (20) are

$$\begin{aligned} r_1(\mu) &= \frac{1}{t}(q^* + 2g^*x + h^*x^2), \\ r_2(\mu) &= \frac{r_1(\mu)}{t}(2(g^* + h^*x) + hr_1(\mu)), \\ r_3(\mu) &= \frac{1}{t}(2(g^* + h^*x)r_2(\mu) + 2hr_1(\mu)r_2(\mu) + h^*r_1^2(\mu)). \end{aligned} \tag{21}$$

If (18) is scaled so that $h = \|H\| = 1$, and

$$z = \max(q^*, g^*, h^*) < 2y(1 + (ty)^{1/2})^{-1}, \qquad y = (1 + x)/t,$$

then according to (20), (21)

$$\|X^*\| \leqq zty^2 + z^2ty^3(2 + y) + z^3ty^4(5 + 6y + 2y^2) + O(z^4), \qquad z \to +0.$$

We conclude the perturbation analysis of matrix quadratic equations with the following remark. The estimate (17), (16) (respectively, (19)) cannot be improved in the sense that for each positive constant $w < 1$ there exists an equation of type (1) (respectively, (18)) such that $\|X^*\| = wr(\mu)$. Moreover, if condition (15) is not fulfilled then the analytic dependence of X^* on A^*, $B_i^*, \cdots, F_i^*$ may not exist, or (5) may not even have a solution.

REFERENCES

[1] G. W. STEWART, *Error and perturbation bounds for subspaces associated with certain eigenvalue problems*, SIAM Rev., 15 (1973), pp. 727-764.

[2] V. KUCERA, *A contribution to the matrix quadratic equations*, IEEE Trans. Automat. Control, 17 (1972), pp. 344-347.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 6, pp. 1164-1180, November 1990

SENSITIVITY ANALYSIS USING THE MONTE CARLO ACCEPTANCE-REJECTION METHOD*

GEORGE S. FISHMAN†

Abstract. This paper describes a Monte Carlo sampling plan for estimating how a function varies in response to changes in its arguments. Most notably, the plan effects this sensitivity analysis by applying the acceptance-rejection technique to data sampled at only one specified setting for the arguments, thus saving considerable computing time when compared to alternative methods. The plan which applies for a 0-1 response on each replication has immediate application when estimating variation in system performance measures in reliability analysis.

The paper derives the variances of the proposed estimators and shows how to use worst-case bounds on these or on corresponding coefficients of variation to choose the arguments, at which to sample, that minimize the worst-case bounds. Individual and simultaneous confidence intervals are derived and an example based on *s-t* reliability illustrates the method. The paper also compares the proposed method and an alternative Monte Carlo approach that uses an importance function.

Key words. acceptance-rejection sampling, Monte Carlo method sensitivity analysis, importance function, reliability

AMS(MOS) subject classifications. 62, 90

Many Monte Carlo sampling experiments aim at estimating quantities of the form

$$g(\mathbf{q})=\sum_{\mathbf{x}\in\mathscr{X}}\phi(\mathbf{x})P(\mathbf{x},\mathbf{q}) \tag{1}$$

where $\{\phi(\mathbf{x})\}$ is a 0-1 binary function, $\{P(\mathbf{x},\mathbf{q}),\mathbf{x}\in\mathscr{X}\}$ is a probability mass function (pmf) with given parameter vector $\mathbf{q}$, and domain of support $\mathscr{X}$ so large as to make exact evaluation via (1) intractable. Occasionally, the objective is to estimate the function $\{g(\mathbf{q}),\mathbf{q}\in\mathscr{Q}\}$ where $\mathscr{Q}=\{\mathbf{q}_1,\cdots,\mathbf{q}_w\}$. Problems of this type arise in reliability theory where $g(\mathbf{q})$ represents system reliability and $\mathbf{q}_j=(q_{j1},\cdots,q_{jr})$ denotes the reliabilities of components of types 1 through r that compose the system in the jth of w component reliability vectors of interest. Analysis of $g(\mathbf{q}_1),\cdots,g(\mathbf{q}_w)$ enables us to assess the benefits of the alternative reliability vectors $\mathbf{q}_1,\cdots,\mathbf{q}_w$ on system reliablity.

Although we can simply run w experiments, sampling from $\{P(\mathbf{x},\mathbf{q}_1)\},\cdots,\{P(\mathbf{x},\mathbf{q}_w)\}$ to produce estimates of $g(\mathbf{q}_1),\cdots,g(\mathbf{q}_w)$, respectively, a more efficient method samples from $\{P(\mathbf{x},\mathbf{p})\}$ on a single experiment and uses these data together with the Monte Carlo importance function technique or the acceptance-rejection technique to produce the desired estimates. These approaches are not new, the importance function technique being implicit in Kahn [8] and Kahn and Harris [9] and the acceptance-rejection technique being implicit in von Neumann [13]. Beckman and McKay [3] have more recently discussed both methods. However, until recently little was known about how the binary property of $\{\phi(\mathbf{x})\}$ affected the sampling properties of these techniques for estimating (1). Fishman [6] provides a comprehensive account of these properties for the importance function approach. The present paper focuses on the acceptance-rejection method and provides a comprehensive description of the sampling properties of the resulting estimators that exploit the binary property of $\{\phi(\mathbf{x})\}$ and the use of a modified pmf $\{Q(\mathbf{x},\mathbf{p})\}$, based on $\{P(\mathbf{x},\mathbf{p})\}$ and information

* Received by the editors September 30, 1988. Accepted for publication (in revised form) October 23, 1989. This research was supported by the Air Force Office of Scientific Research under grant AFOSR-84-0140.

† Department of Operations Research, University of North Carolina, Chapel Hill, North Carolina 27599.

on bounds for $\{\phi(\mathbf{x})\}$ and $\{g(\mathbf{q}), \mathbf{q} \in \mathcal{Q}\}$, to sample the data. This last modification allows the acceptance-rejection method to work with considerably improved efficiency. Although the paper focuses on applying the proposed technique to reliability estimation, we emphasize that the methodology applies to the considerably wider class of problems with binary $\{\phi(\mathbf{x})\}$.

Section 1 gives basic definitions and § 2 describes estimation at a single point. Section 3 then describes how to perform function estimation using the acceptance-rejection method. Section 4 shows how to choose the design parameter $\mathbf{p}$ to minimize either the worst-case variance or the coefficient of variation of the resulting function estimator, thereby dramatically increasing the efficiency of the proposed Monte Carlo procedure. Section 5 shows that even in the worst case, the proposed technique is at least as good as crude Monte Carlo sampling. Sections 6 and 7 derive individual and simultaneous confidence intervals. Section 8 illustrates the proposed technique with an example and § 9 compares the characteristics of the acceptance-rejection method with those of the importance function method.

1. Problem setting. Consider a network $G = (\mathcal{V}, \mathcal{E})$ with node set $\mathcal{V}$ and edge set $\mathcal{E}$. Assume that nodes function perfectly and that edges fail randomly and independently. Let

r = number of distinct types of edges,

q_i = probability that an edge of type i functions $i = 1, \cdots, r$,

$\mathbf{q} = (q_1, \cdots, q_r)$,

k_i = number of edges of type i,

$\mathbf{k} = (k_1, \cdots, k_r)$,

e_{ij} = jth edge of type i, $j = 1, \cdots, k_i$, $i = 1, \cdots, r$,

(2) $x_{ij} = 1$ if edge e_{ij} functions
$\quad = 0$ otherwise,

$x_i = \sum_{j=1}^{k_i} x_{ij}$ = number of functioning edges of type i,

$\mathbf{x} = (x_{11}, \cdots, x_{1k_1}; \cdots; x_{r1}, \cdots, x_{rk_r})$,

$\mathcal{X}$ = set of all edge states $\mathbf{x}$,

$$P(\mathbf{x}, \mathbf{q}) = P(\mathbf{x}, \mathbf{k}, \mathbf{q}) = \prod_{i=1}^{r} \prod_{j=1}^{k_i} [x_{ij}q_i + (1 - x_{ij})(1 - q_i)] = \prod_{i=1}^{r} q_i^{x_i}(1 - q_i)^{k_i - x_i}$$

$\quad$ = pmf of state $\mathbf{x} \in \mathcal{X}$,

$\phi(\mathbf{x}) = 1$ if the system functions when in state $\mathbf{x}$
$\quad = 0$ otherwise,

$$g(\mathbf{q}) = \sum_{\mathbf{x} \in \mathcal{X}} \phi(\mathbf{x}) P(\mathbf{x}, \mathbf{q}) \tag{3}$$

$\quad$ = probability that the system functions.

We also assume that G describes a *coherent* system. A system of components is coherent if its structure function $\{\phi(\mathbf{x})\}$ is nondecreasing in each argument and each component is relevant (Barlow and Proschan [2, p. 6]). Let $\mathcal{Q}$ denote a set of $w = |\mathcal{Q}|$ component reliability vectors of interest. Then the purpose of analysis is to estimate the reliability function $\{g(\mathbf{q}), \mathbf{q} \in \mathcal{Q}\}$.

2. Estimation at a point. Crude Monte Carlo sampling offers a baseline against which potentially more efficient sampling plans can be compared. Let $\mathbf{X}^{(1)}, \cdots, \mathbf{X}^{(K)}$ denote K independent samples drawn from $\{P(\mathbf{x}, \mathbf{q}), \mathbf{x} \in \mathscr{X}\}$. Then

$$\bar{g}_K(\mathbf{q}) = \frac{1}{K} \sum_{i=1}^{K} \phi(\mathbf{X}^{(i)}) \tag{4}$$

is an unbiased estimator of $g(\mathbf{q})$ with

$$\operatorname{var} \bar{g}_K(\mathbf{q}) = g(\mathbf{q})[1 - g(\mathbf{q})]/K. \tag{5}$$

To compute $\bar{g}_K(\mathbf{q})$, we perform K trials sampling $\mathbf{X}$ from $\{P(\mathbf{x}, \mathbf{k}, \mathbf{q})\}$ and evaluate $\phi(\mathbf{X})$ on each trial. The corresponding mean total computation time has the form

$$T(\bar{g}_K(\mathbf{q})) = \alpha_0 + K[\alpha_1 + \alpha_2 |\mathscr{E}| + \alpha_3(\mathscr{X}, \mathbf{q})]$$

where

$$\alpha_3(\mathscr{X}, \mathbf{q}) = \sum_{\mathbf{x} \in \mathscr{X}} P(\mathbf{x}, \mathbf{q}) C(\mathbf{x})$$

and

$$C(\mathbf{x}) = \text{expected time to evaluate } \phi(\mathbf{x}).$$

The quantities α_0, α_1, α_2 and $\alpha_3(\mathscr{X}, \mathbf{q})$ are machine dependent.

We now show how to modify the sampling plan to improve statistical efficiency using information on bounds as described in Fishman [5]. Suppose that there exist 0-1 binary functions $\{\phi_L(\mathbf{x}), \mathbf{x} \in \mathscr{X}\}$ and $\{\phi_U(\mathbf{x}), \mathbf{x} \in \mathscr{X}\}$ such that

$$\phi_L(\mathbf{x}) \leqq \phi(\mathbf{x}) \leqq \phi_U(\mathbf{x}) \quad \forall \mathbf{x} \in \mathscr{X}.$$

Then $g(\mathbf{q})$ has lower and upper bounds $g_L(\mathbf{q})$ and $g_U(\mathbf{q})$, respectively, where

$$g_i(\mathbf{q}) = \sum_{\mathbf{x} \in \mathscr{X}} \phi_i(\mathbf{x}) P(\mathbf{x}, \mathbf{q}), \qquad i \in \{L, U\}.$$

Suppose that we now sample $\mathbf{X}^{(1)}, \cdots, \mathbf{X}^{(K)}$ independently from the modified pmf

$$Q(\mathbf{x}, \mathbf{q}) = \left[\frac{\phi_U(\mathbf{x}) - \phi_L(\mathbf{x})}{\Delta(\mathbf{q})}\right] P(\mathbf{x}, \mathbf{q}), \qquad \mathbf{x} \in \mathscr{X} \tag{6}$$

where

$$\Delta(\mathbf{q}) = g_U(\mathbf{q}) - g_L(\mathbf{q}).$$

Then

$$\hat{g}_K(\mathbf{q}) = g_L(\mathbf{q}) + \Delta(\mathbf{q}) \frac{1}{K} \sum_{i=1}^{K} \phi(\mathbf{X}^{(i)}) \tag{7}$$

is also an unbiased estimator of $g(\mathbf{q})$, but with variance

$$\operatorname{var} \hat{g}_K(\mathbf{q}) = [g_U(\mathbf{q}) - g(\mathbf{q})][g(\mathbf{q}) - g_L(\mathbf{q})]/K \leqq \Delta^2(\mathbf{q})/4K. \tag{8}$$

Compared to crude Monte Carlo sampling, we have

$$\begin{aligned} \frac{\operatorname{var} \bar{g}_K(\mathbf{q})}{\operatorname{var} \hat{g}_K(\mathbf{q})} &\geqq D(\mathbf{q}) = 1/[\{g_L(\mathbf{q})[1 - g_U(\mathbf{q})]\}^{1/2} - \{g_U(\mathbf{q})[1 - g_L(\mathbf{q})]\}^{1/2}]^2 \\ &\geqq 1, \end{aligned} \tag{9}$$

indicating that $\hat{g}_K(\mathbf{q})$ always has a variance no larger than $\operatorname{var} \bar{g}_K(\mathbf{q})$.

To compute $\hat{g}_K(\mathbf{q})$ using precomputed bounds, we perform K trials sampling $\mathbf{X}$ from $\{Q(\mathbf{x}, \mathbf{q})\}$ and evaluate $\phi(\mathbf{X})$ on each trial. Here mean total time assumes the form

$$T(\hat{g}_K(\mathbf{q})) = \beta_0 + K[\beta_1 + \beta_2|\mathscr{C}| + \alpha_3(\mathscr{X}_{01}, \mathbf{p})/\Delta(\mathbf{q})]$$

where

$$\mathscr{X}_{01} = \{\mathbf{x} \in \mathscr{X}\colon \phi_L(\mathbf{x}) = 0 \text{ and } \phi_U(\mathbf{x}) = 1\}$$

and β_0, β_1, and β_2 denote machine-dependent constants.

Observe that

$$K(\mathbf{q}) = K \operatorname{var} \bar{g}_K(\mathbf{q})/\operatorname{var} \hat{g}_K(\mathbf{q})$$

denotes the number of observations we would have to take with crude Monte Carlo to achieve the same variance that arises in K observations using $\{Q(\mathbf{x}, \mathbf{q})\}$. Then $\Lambda_1(\mathbf{q}) = T(\bar{g}_{K(q)}(\mathbf{q}))/T(\hat{g}_K(\mathbf{q}))$ measures the efficiency of $\hat{g}_K(\mathbf{q})$ relative to $\bar{g}_K(\mathbf{q})$ and for large K and $|\mathscr{C}|$ has the approximate form

(10)
$$\begin{aligned}\Lambda_1(\mathbf{q}) &\approx \left[\frac{\alpha_2 + \alpha_3(\mathscr{X}, \mathbf{q})/|\mathscr{C}|}{\beta_2 + \alpha_3(\mathscr{X}_{01}, \mathbf{q})/\Delta(\mathbf{q})|\mathscr{C}|}\right] \frac{g(\mathbf{q})[1 - g(\mathbf{q})]}{[g_U(\mathbf{q}) - g(\mathbf{q})][g(\mathbf{q}) - g_L(\mathbf{q})]} \\ &\geqq \left[\frac{\alpha_2 + \alpha_3(\mathscr{X}, \mathbf{q})/|\mathscr{C}|}{\beta_2 + \alpha_3(\mathscr{X}_{01}, \mathbf{q})/\Delta(\mathbf{q})|\mathscr{C}|}\right] D(\mathbf{q})\end{aligned}$$

where (9) defines $D(\mathbf{q}) \geqq 1$. A ratio greater than unity favors the alternative sampling plan. Experience (Fishman [5]) has shown this to be the case for moderate and high component reliabilities for s-t reliability.

3. Function estimation based on the acceptance-rejection method.

To estimate $g(\mathbf{q})$ for each component reliability vector $\mathbf{q} \in \mathscr{Q} = \{\mathbf{q}_1, \cdots, \mathbf{q}_w\}$, we can perform w separate experiments, sampling from $\{Q(\mathbf{x}, \mathbf{q}_i)\}$ in (6) on the ith experiment for $i = 1, \cdots, w$. This procedure incurs the cost of running w individual sampling experiments. However, we can actually avoid this cost by performing a single experiment, sampling data from $\{Q(\mathbf{x}, \mathbf{p})\}$, and then using these same data to estimate $g(\mathbf{q}_1), \cdots, g(\mathbf{q}_w)$. We later show that if the component reliablity vector $\mathbf{p}$ at which sampling occurs belongs to $\mathscr{Q}$, the proposed approach leads to estimates of specified accuracy at a cost no larger than that incurred by performing all w individual experiments to achieve the identical accuracies.

Consider the pmf

(11)
$$f(\mathbf{x}) = ab(\mathbf{x})c(\mathbf{x}), \qquad \mathbf{x} \in \mathscr{X}$$

where

$$c(\mathbf{x}) \geqq 0, \quad \sum_{\mathbf{x} \in \mathscr{X}} c(\mathbf{x}) = 1, \quad 0 \leqq b(\mathbf{x}) \leqq 1, \quad a = 1/\sum_{\mathbf{x} \in \mathscr{X}} b(\mathbf{x})c(\mathbf{x}).$$

Suppose we sample $\mathbf{X}$ from the pmf $\{c(\mathbf{x})\}$ and Z from $\mathscr{U}(0, 1)$. If $Z \leqq b(\mathbf{X})$, then $\mathbf{X}$ has the pmf $\{f(\mathbf{x})\}$ in (11). This acceptance-rejection method of sampling is due to von Neumann [13]. For the current problem,

(12)
$$c(\mathbf{x}) = Q(\mathbf{x}, \mathbf{p}), \qquad b(\mathbf{x}) = \frac{R(\mathbf{x}, \mathbf{q}, \mathbf{p})}{R^*(\mathbf{q}, \mathbf{p})},$$

where

$$R(\mathbf{x}, \mathbf{q}, \mathbf{p}) = \frac{P(\mathbf{x}, \mathbf{q})}{P(\mathbf{x}, \mathbf{p})} \tag{13}$$
$$= \prod_{i=1}^{r} \left(\frac{q_i}{p_i}\right)^{x_i} \left(\frac{1-q_i}{1-p_i}\right)^{k_i - x_i}$$

and

$$R^*(\mathbf{q}, \mathbf{p}) = \max_{\mathbf{x} \in \mathscr{X}} R(\mathbf{x}, \mathbf{q}, \mathbf{p}) \quad \text{subject to } \phi_L(\mathbf{x}) = 0 \text{ and } \phi_U(\mathbf{x}) = 1.$$

The quantity

$$a = \frac{\Delta(\mathbf{p}) R^*(\mathbf{q}, \mathbf{p})}{\Delta(\mathbf{q})} \tag{14}$$

denotes the mean number of trials required until we successfully obtain an **X** from $\{Q(\mathbf{x}, \mathbf{q})\}$.

A small modification increases the efficiency of this procedure. Let

$$R_0(\mathbf{q}, \mathbf{p}) = \max_{\mathbf{x} \in \mathscr{X}} R(\mathbf{x}, \mathbf{q}, \mathbf{p}) \quad \text{subject to } \phi(\mathbf{x}) = 0 \text{ and } \phi_U(\mathbf{x}) = 1, \tag{15a}$$

$$R_1(\mathbf{q}, \mathbf{p}) = \max_{\mathbf{x} \in \mathscr{X}} R(\mathbf{x}, \mathbf{q}, \mathbf{p}) \quad \text{subject to } \phi_L(\mathbf{x}) = 0 \text{ and } \phi(\mathbf{x}) = 1, \tag{15b}$$

and

$$F(\mathbf{x}, i, \mathbf{q}, \mathbf{p}) = \frac{R(\mathbf{x}, \mathbf{q}, \mathbf{p})}{R_i(\mathbf{q}, \mathbf{p})}, \qquad i = 0, 1. \tag{16}$$

Suppose we sample **X** from $\{Q(\mathbf{x}, \mathbf{p})\}$, sample Z from $\mathscr{U}(0, 1)$, and determine $\phi(\mathbf{X})$. If $Z \leqq F(\mathbf{X}, \phi(\mathbf{X}), \mathbf{q}, \mathbf{p})$, then **X** has the pmf $\{Q(\mathbf{x}, \mathbf{q})\}$ with mean number of trials until success

$$a = \frac{g_U(\mathbf{p}) - g(\mathbf{p})}{\Delta(\mathbf{q})} R_0(\mathbf{q}, \mathbf{p}) + \frac{g(\mathbf{p}) - g_L(\mathbf{p})}{\Delta(\mathbf{q})} R_1(\mathbf{q}, \mathbf{p})$$
$$\leqq \frac{\Delta(\mathbf{p}) R^*(\mathbf{q}, \mathbf{p})}{\Delta(\mathbf{q})},$$

since $\max\,[g_U(\mathbf{p}) - g(\mathbf{p}), g(\mathbf{p}) - g_L(\mathbf{p})] \leqq \Delta(\mathbf{p})$ and $\max\,[R_0(\mathbf{q}, \mathbf{p}), R_1(\mathbf{q}, \mathbf{p})] \leqq R^*(\mathbf{q}, \mathbf{p})$. The computations of $R_0(\mathbf{q}, \mathbf{p})$ and $R_1(\mathbf{q}, \mathbf{p})$ depend on the choice of bounding functions $\{\phi_L(\mathbf{x})\}$ and $\{\phi_U(\mathbf{x})\}$ and are discussed in the example in § 8.

We next describe the statistical properties of data generated by the acceptance-rejection method.

THEOREM 1. *Let* **X** *and* Z *denote samples drawn from* $\{Q(\mathbf{x}, \mathbf{p})\}$ *in* (6) *and* $\mathscr{U}(0, 1)$, *respectively. Define* $R_0 \equiv R_0(\mathbf{q}, \mathbf{p})$, $R_1 \equiv R_1(\mathbf{q}, \mathbf{p})$,

$$\begin{aligned} \theta_i(\mathbf{x}, u, \mathbf{q}, \mathbf{p}) &= 1 \quad \text{if } 0 \leqq u \leqq R(\mathbf{x}, \mathbf{q}, \mathbf{p})/R_i, \quad i = 0, 1, \\ &= 0 \quad \textit{otherwise}, \end{aligned} \tag{17}$$

$$\mu_0(\mathbf{x}, u, \mathbf{q}, \mathbf{p}) = g_U(\mathbf{q}) - \Delta(\mathbf{p}) R_0[1 - \phi(\mathbf{x})] \theta_{\phi(\mathbf{x})}(\mathbf{x}, u, \mathbf{q}, \mathbf{p}), \tag{18a}$$

and

$$\mu_1(\mathbf{x}, u, \mathbf{q}, \mathbf{p}) = g_L(\mathbf{q}) + \Delta(\mathbf{p}) R_1 \phi(\mathbf{x}) \theta_{\phi(\mathbf{x})}(\mathbf{x}, u, \mathbf{q}, \mathbf{p}). \tag{18b}$$

Then

(i) $E\{[1-\phi(\mathbf{X})]\theta_{\phi(\mathbf{x})}(\mathbf{X},Z,\mathbf{q},\mathbf{p})\}=[g_U(\mathbf{q})-g(\mathbf{q})]/\Delta(\mathbf{p})R_0,$

(ii) $E[\phi(\mathbf{X})\theta_{\phi(\mathbf{x})}(\mathbf{X},Z,\mathbf{q},\mathbf{p})]=[g(\mathbf{q})-g_L(\mathbf{q})]/\Delta(\mathbf{p})R_1,$

(iii) $E\mu_i(\mathbf{X},Z,\mathbf{q},\mathbf{p})=g(\mathbf{q}), \qquad \mathrm{i}=0,1,$

(iv) $\operatorname{var}\mu_0(\mathbf{X},Z,\mathbf{q},\mathbf{p})=[g_U(\mathbf{q})-g(\mathbf{q})][g(\mathbf{q})-g_U(\mathbf{q})+\Delta(\mathbf{p})R_0]$
$=v(\mathbf{q})+[g_U(\mathbf{q})-g(\mathbf{q})][\Delta(\mathbf{p})R_0-\Delta(\mathbf{q})],$

(v) $\operatorname{var}\mu_1(\mathbf{X},Z,\mathbf{q},\mathbf{p})=[g(\mathbf{q})-g_L(\mathbf{q})][\Delta(\mathbf{p})R_1+g_L(\mathbf{q})-g(\mathbf{q})]$
$=\mathrm{v}(\mathbf{q})+[g(\mathbf{q})-g_L(\mathbf{q})][\Delta(\mathbf{p})R_1-\Delta(\mathbf{q})],$

(vi) $\operatorname{cov}[\mu_0(\mathbf{X},Z,\mathbf{q},\mathbf{p}),\mu_1(\mathbf{X},Z,\mathbf{q},\mathbf{p})]=v(\mathbf{q})=[g_U(\mathbf{q})-g(\mathbf{q})][g(\mathbf{q})-g_L(\mathbf{q})].$

Proof. Straightforwardly,

$$\begin{aligned}E\{[1-\phi(\mathbf{X})]\theta_{\phi(\mathbf{x})}(\mathbf{X},Z,\mathbf{q},\mathbf{p})\}&=\operatorname{pr}[\phi(\mathbf{X})=0,\ \theta_0(\mathbf{X},Z,\mathbf{q},\mathbf{p})=1]\\&=\sum_{\mathbf{x}\in\mathscr{X}}\left\{[1-\phi(\mathbf{x})]\frac{R(\mathbf{x},\mathbf{q},\mathbf{p})}{R_{\phi(\mathbf{x})}}\right\}\frac{[\phi_U(\mathbf{x})-\phi_L(\mathbf{x})]}{\Delta(\mathbf{p})}P(\mathbf{x},\mathbf{p})\\&=\frac{[g_U(\mathbf{q})-g(\mathbf{q})]}{\Delta(\mathbf{p})}R_0,\end{aligned}$$

establishing (i). Part (ii) follows in analogous fashion and the proofs of parts (iii)-(vi) are then immediately obvious.

Suppose we perform K independent replications generating $\mathbf{X}^{(1)},\cdots,\mathbf{X}^{(K)}$ from (6) and $Z^{(1)},\cdots,Z^{(K)}$ from $\mathscr{U}(0,1)$. Then

$$\tilde{g}_{iK}(\mathbf{q},\mathbf{p})=\frac{1}{K}\sum_{j=1}^{K}\mu_i(\mathbf{X}^{(j)},Z^{(j)},q,p), \qquad i=0,1 \tag{19}$$

have expectations $g(\mathbf{q})$ with $\operatorname{var}\hat{g}_{iK}(\mathbf{q},\mathbf{p})=\operatorname{var}\mu_i(\mathbf{X},Z,\mathbf{q},\mathbf{p})/K$. Observe that the inequalities $\operatorname{var}\mu_0(\mathbf{X},Z,\mathbf{q},\mathbf{p})>v(\mathbf{q})$ and $\operatorname{var}\mu_1(\mathbf{X},Z,\mathbf{q},\mathbf{p})]>v(\mathbf{q})$ for $\mathbf{q}\neq\mathbf{p}$, when they occur, signal an inflation of variances over what is obtained if we were to sample from $\{Q(\mathbf{x},\mathbf{q})\}$ directly. Therefore, it is of interest to assess how much these variances and corresponding coefficients of variation grow when using the proposed acceptance-rejection method. Theorems 2 and 3 provide worst-case upper bounds.

THEOREM 2. *Let* $\mathbf{X}$ *and* Z *denote samples from* $\{Q(\mathbf{x},\mathbf{p})\}$ *in* (6) *and* $\mathscr{U}(0,1)$, *respectively. Then*

$$\operatorname{var}\mu_i(\mathbf{X},Z,\mathbf{q},\mathbf{p})\leqq M_i(\mathbf{q},\mathbf{p})=\begin{cases}[\Delta(\mathbf{p})R_i]^2/4 & \text{if } \Delta(\mathbf{q})>\Delta(\mathbf{p})R_i/2,\\ \Delta(\mathbf{q})[\Delta(\mathbf{p})R_i-\Delta(\mathbf{q})] & \text{if } \Delta(\mathbf{q})\leqq\Delta(\mathbf{p})R_i/2,\end{cases} \tag{20}$$
$$i=0,1.$$

Proof. Since $g_L(\mathbf{q})\leqq g(\mathbf{q})\leqq g_U(\mathbf{q})$, $A=[g_U(\mathbf{q})-g(\mathbf{q})][g(\mathbf{q})-g_U(\mathbf{q})+\Delta(\mathbf{p})R_0]$ has its maximum at $g^*(\mathbf{q})=g_L(\mathbf{q})+\max[0,\Delta(\mathbf{q})-\Delta(\mathbf{p})R_0/2]$, from which (20) follows for $i=0$. Similarly, $B=[g(\mathbf{q})-g_L(\mathbf{q})][\Delta(\mathbf{p})R_1+g_L(\mathbf{q})-g(\mathbf{q})]$ has its maximum at $g^*(\mathbf{q})=g_U(\mathbf{q})-\max[0,\Delta(\mathbf{q})-\Delta(\mathbf{p})R_1/2]$, from which (20) follows for $i=1$.

Observe that evaluation of (20) for $i=0$, 1, prior to sampling, enables us to determine which estimator has the smallest worst-case variance.

THEOREM 3. *Let*

$$\gamma_i(\mathbf{q},\mathbf{p})=[\operatorname{var}\mu_i(\mathbf{X},Z,\mathbf{q},\mathbf{p})]^{1/2}/[1-\mathrm{g}(\mathbf{q})], \qquad i=0,1.$$

Then

$$\max_{g_L(\mathbf{q})\leqq g(\mathbf{q})\leqq g_U(\mathbf{q})} \gamma_0^2(\mathbf{q},\mathbf{p}) = N_0(\mathbf{q},\mathbf{p}) = [\Delta(\mathbf{p})R_0]^2/4[1-g_U(\mathbf{q})+\Delta(\mathbf{p})R_0][1-g_U(\mathbf{q})] \tag{21a}$$

$$\textit{if } \Delta(\mathbf{p})R_0[1-g_U(\mathbf{q})-\Delta(\mathbf{q})]\leqq 2\Delta(\mathbf{q})[1-g_U(\mathbf{q})]$$

$$= \Delta(\mathbf{q})[\Delta(\mathbf{p})R_0-\Delta(\mathbf{q})]/[1-g_L(\mathbf{q})]^2 \quad \textit{otherwise} \tag{21b}$$

and

$$\max_{g_L(\mathbf{q})\leqq g(\mathbf{q})\leqq g_U(\mathbf{q})} \gamma_1^2(\mathbf{q},\mathbf{p}) = N_1(\mathbf{q},\mathbf{p}) = [\Delta(\mathbf{p})R_1]^2/4[1-g_L(\mathbf{q})-\Delta(\mathbf{p})R_1][1-g_L(\mathbf{q})] \tag{22a}$$

$$\textit{if } \Delta(\mathbf{p})R_1[1-g_L(\mathbf{q})+\Delta(\mathbf{q})]\leqq 2\Delta(\mathbf{q})[1-g_L(\mathbf{q})]$$

$$= \Delta(\mathbf{q})[\Delta(\mathbf{p})R_1-\Delta(\mathbf{q})]/[1-g_U(\mathbf{q})]^2 \quad \textit{otherwise.} \tag{22b}$$

Proof. We give the proof for max $\gamma_0^2(\mathbf{q},\mathbf{p})$. Let

$$A = \operatorname{var} \mu_0(\mathbf{X}, Z, \mathbf{q}, \mathbf{p}) \ /[1-g(\mathbf{q})]^2. \tag{23}$$

Then

$$\frac{\partial A}{\partial g} = \frac{-g(\mathbf{q})\{2[1-g_U(\mathbf{q})]+\Delta(\mathbf{p})R_0\}+2g_U(\mathbf{q})[1-g_U(\mathbf{q})]-\Delta(\mathbf{p})R_0[1-2g_U(\mathbf{q})]}{[1-g(\mathbf{q})]^3}.$$

Since $\partial A/\partial g(\mathbf{q})|_{g(\mathbf{q})=g_U(\mathbf{q})} < 0$ and $\partial A/\partial g(\mathbf{q}) = 0$ at

$$g^*(\mathbf{q}) = \{2g_U(\mathbf{q})[1-g_U(\mathbf{q})]-\Delta(\mathbf{p})R_0[1-2g_U(\mathbf{q})]\}/\{2[1-g_U(\mathbf{q})]+\Delta(\mathbf{p})R_0\},$$

A has its maximum at $g^*(\mathbf{q})$ if $g^*(\mathbf{q}) \geqq g_L(\mathbf{q})$, which upon substitution of $g^*(\mathbf{q})$ for $g(\mathbf{q})$ in (23) gives (21a). If $g^*(\mathbf{q}) < g_L(\mathbf{q})$ then the maximum occurs at $g_L(\mathbf{q})$, giving (21b). A completely analogous result holds for max $\gamma_1^2(\mathbf{q},\mathbf{p})$.

4. Choosing the sampling probabilities *p*. The results in Theorems 2 and 3 play a critical role in deciding at which component reliability vector $\mathbf{p}$ we should conduct the Monte Carlo sampling experiment. For each $i = 0, 1$, one procedure finds the $\mathbf{p}_i \in \mathcal{Q}$ that minimizes $\max_{\mathbf{q}\in\mathcal{Q}} M_i(\mathbf{q},\mathbf{p})$ where (20) defines $M_i(\mathbf{q},\mathbf{p})$ as the worst-case var $\mu_i(\mathbf{X}, Z, \mathbf{q}, \mathbf{p})$. Then we use

$$\begin{aligned} \mathbf{p} &= \mathbf{p}_0 \quad \text{if} \max_{\mathbf{q}\in\mathcal{Q}} M_0(\mathbf{q},\mathbf{p}_0) \leqq \max_{\mathbf{q}\in\mathcal{Q}} M_1(\mathbf{q},\mathbf{p}_1),\\ &= \mathbf{p}_1 \quad \text{otherwise,}\end{aligned} \tag{24}$$

so that sampling from $\{Q(\mathbf{x},\mathbf{p})\}$ with $\mathbf{p}$ as in (24) minimizes the worst-case variance that can arise. Finding $\mathbf{p}_i$ takes w^2 evaluations of $M_i(\mathbf{q},\mathbf{p})$. Also, note that

$$K_* = \lceil \min [\max_{\mathbf{q}\in\mathcal{Q}} M_0(\mathbf{q},\mathbf{p}_0), \max_{\mathbf{q}\in\mathcal{Q}} M_1(\mathbf{q},\mathbf{p}_1)]/v_* \rceil$$

gives the worst-case sample size required to obtain estimates of $g(\mathbf{q}_1), \cdots, g(\mathbf{q}_w)$ with variances no greater than a specified v_*. This valuable information can assist a user of the Monte Carlo method before any sampling begins.

The proposed technique can also accommodate a relative accuracy specification. For $i = 0, 1$, an alternative procedure finds the $\mathbf{p}_i \in \mathcal{Q}$ that minimizes $\max_{\mathbf{q}\in\mathcal{Q}} N_i(\mathbf{q},\mathbf{p})$ where (21) and (22) define $N_0(\mathbf{q},\mathbf{p})$ and $N_1(\mathbf{q},\mathbf{p})$, and then uses

$$\begin{aligned} \mathbf{p} &= \mathbf{p}_0 \quad \text{if} \max_{\mathbf{q}\in\mathcal{Q}} N_0(\mathbf{q},\mathbf{p}_0) \leqq \max_{\mathbf{q}\in\mathcal{Q}} N_1(\mathbf{q},\mathbf{p}_1)\\ &= \mathbf{p}_1 \quad \text{otherwise.}\end{aligned}$$

Sampling from $\{Q(\mathbf{x}, \mathbf{p})\}$ with this $\mathbf{p}$ minimizes the worst-case coefficient of variation. Also,

$$K_{**} = \lceil \min [\max_{\mathbf{q}\in\mathcal{Q}} N_0(\mathbf{q}, \mathbf{p}_0), \max_{\mathbf{q}\in\mathcal{Q}} N_1(\mathbf{q}, \mathbf{p}_1)]/u_*^2 \rceil \tag{25}$$

provides the worst-case sample size needed to estimate $g(\mathbf{q}_1), \cdots, g(\mathbf{q}_w)$ with coefficients of variation no greater than a specified u_*.

5. Efficiency. Naturally, the appeal of any proposed sampling plan depends on the cost saving it offers, when achieving a specified accuracy as compared to other more conventional methods. These cost considerations have two components, one based on variances and the other based on computer times expended per replication. Theorem 4 derives an expression for the smallest variance ratio that we can expect to achieve when comparing a crude Monte Carlo estimate $\bar{g}_K(\mathbf{q})$ to an estimate $\tilde{g}_{iK}(\mathbf{q}, \mathbf{p})$ based on the proposed method. This smallest ratio is analogous to $D(\mathbf{q})$ in (9) and reveals the least favorable circumstance that we can expect to encounter. The ratio can be computed prior to sampling, thereby providing a lower bound on what to expect.

THEOREM 4. *Let* $\mathbf{Y}$ *denote a sample from* $\{P(\mathbf{x}, \mathbf{p})\}$, $\mathbf{X}$ *a sample drawn from* $\{Q(\mathbf{x}, \mathbf{p})\}$ *and* Z *a sample drawn from* $\mathcal{U}(0, 1)$. *Let*

$$B_i(g(\mathbf{q}), \mathbf{p}) = \operatorname{var} \phi(\mathbf{Y})/\operatorname{var} \mu_i(\mathbf{X}, Z, \mathbf{q}, \mathbf{p}), \qquad i = 0, 1.$$

Then

$$\begin{aligned}\min_{g_L(\mathbf{q})\leqq g(\mathbf{q})\leqq g_U(\mathbf{q})} B_0(g(\mathbf{q}), \mathbf{p}) &= 1/\{\{g_U(\mathbf{q})[1-g_U(\mathbf{q})+\Delta(\mathbf{p})R_0]\}^{1/2} \\ &\quad -\{[1-g_U(\mathbf{q})][g_U(\mathbf{q})-\Delta(\mathbf{p})R_0]\}^{1/2}\}^2 \\ &\quad \textit{if } \Delta(\mathbf{p})R_0 \leqq \frac{\Delta(\mathbf{q})\{g_L(\mathbf{q})[1-g_U(\mathbf{q})]+g_U(\mathbf{q})[1-g_L(\mathbf{q})]\}}{\Delta^2(\mathbf{q})+g_U(\mathbf{q})[1-g_U(\mathbf{q})]} \\ &= g_L(\mathbf{q})[1-g_L(\mathbf{q})]/\Delta(\mathbf{q})[\Delta(\mathbf{p})R_0-\Delta(\mathbf{q})] \quad \textit{otherwise}\end{aligned} \tag{26}$$

and

$$\begin{aligned}\min_{g_L(\mathbf{q})\leqq g(\mathbf{q})\leqq g_U(\mathbf{q})} B_1(g(\mathbf{q}), \mathbf{p}) &= 1/\{\{g_L(\mathbf{q})[1-g_L(\mathbf{q})-\Delta(\mathbf{p})R_1]\}^{1/2} \\ &\quad -\{[1-g_L(\mathbf{q})][g_L(\mathbf{q})+\Delta(\mathbf{p})R_1]\}^{1/2}\}^2 \\ &\quad \textit{if } \Delta(\mathbf{p})R_1 \leqq \frac{\Delta(\mathbf{q})\{g_L(\mathbf{q})[1-g_U(\mathbf{q})]+g_U(\mathbf{q})[1-g_L(\mathbf{q})]\}}{\Delta^2(\mathbf{q})+g_L(\mathbf{q})[1-g_L(\mathbf{q})]} \\ &= g_U(\mathbf{q})[1-g_U(\mathbf{q})]/\Delta(\mathbf{q})[\Delta(\mathbf{p})R_1-\Delta(\mathbf{q})] \quad \textit{otherwise.}\end{aligned} \tag{27}$$

Proof. We prove the result for $B_1(g(\mathbf{q}), \mathbf{p})$. Observe that $\partial B_1/\partial g(\mathbf{q}) = 0$ has roots

$$r_i = 1\Bigg/\left\{1+(-1)^i\left[\frac{1-g_L(\mathbf{q})}{g_L(\mathbf{q})}\cdot\left(\frac{1-g_L(\mathbf{q})-\Delta(\mathbf{p})R_1}{g_L(\mathbf{q})+\Delta(\mathbf{p})R_1}\right)\right]^{1/2}\right\}, \qquad i = 1, 2. \tag{28}$$

If $\Delta(\mathbf{p})R_1 \leqq 1-g_L(\mathbf{q})$, the roots are real with either $r_1 \leqq 0$ or $r_1 \geqq 1$ and $g_L(\mathbf{q}) \leqq r_2 \leqq 1$. Since $\partial B_1/\partial g(\mathbf{q})|_{g(\mathbf{q})=g_U(\mathbf{q})} < 0$, then

$$\begin{aligned}\min_{g_L(\mathbf{q})\leqq g(\mathbf{q})\leqq g_U(\mathbf{q})} B_1(g(\mathbf{q}), \mathbf{p}) &= B_1(r_2, \mathbf{p}) \quad \text{if } r_2 \leqq g_U(\mathbf{q}) \\ &= B_1(g_U(\mathbf{q}), \mathbf{p}) \quad \text{if } r_2 \geqq g_U(\mathbf{q}).\end{aligned}$$

Expression (27) follows from substituting (28) for r_2 in the inequality. If $\Delta(\mathbf{p})R_1 > 1-g_L(\mathbf{q})$, then the roots are complex and $\partial B_1/\partial g(\mathbf{q}) < 0$ for all $g(\mathbf{q}) \in [g_L(\mathbf{q}), g_U(\mathbf{q})]$ so that the minimum occurs at $g(\mathbf{q})=g_U(\mathbf{q})$. Moreover, complex roots imply that the condition in the upper branch of (27) is always true, thus completing the proof. An analogous result holds for $B_0(g(\mathbf{q}),\mathbf{p})$. □

The availablity of (26) and (27) for each $\mathbf{q}$ in $\mathcal{Q}$ again provides valuable information to the Monte Carlo user prior to experimentation. In particular, it identifies at which $\mathbf{q}$ adverse variance ratios may occur. However, measuring the statistical efficiency of $\{\tilde{g}_{0K}(\mathbf{q},\mathbf{p}),\mathbf{q}\in\mathcal{Q}\}$ and $\{\tilde{g}_{1K}(\mathbf{q},\mathbf{p}),\mathbf{q}\in\mathcal{Q}\}$ as estimators of $\{g(\mathbf{q}),\mathbf{q}\in\mathcal{Q}\}$ calls for a more elaborate analysis than that for estimation at a single point. In particular, the sobering observation that R_0 and R_1 in (26) and (27) increase exponentially with $|\mathscr{E}|$ makes one circumspect about the benefit of the proposed method as the size of G grows. We now show that this benefit is assured for finite $w=|\mathcal{Q}|$ and number of edge types r, provided that $\mathbf{p}\in\mathcal{Q}$.

Recall that $\mathcal{Q}=\{\mathbf{q}_1,\cdots,\mathbf{q}_w\}$ where $\mathbf{q}_j=(q_{1j},\cdots,q_{rj})$ and q_{ij} is the reliability assigned to components of type i in the jth component reliability vector for $j=1,\cdots,w$. Let $\mathscr{H}=\{1,\cdots,r\}$ and

$$\mathscr{H}^*=\{i\in\mathscr{H}\colon p_i\neq q_{ij} \text{ for at least one } j;\ j=1,\cdots,w\},$$

so that $|\mathscr{H}^*|$ component reliablity types vary in $\mathcal{Q}$.

Algorithm A-R describes the steps for computing the estimates and provides the basis for measuring efficiency:

ALGORITHM A-R.

Purpose: to estimate the reliability function $\{g(\mathbf{q}),\mathbf{q}\in\mathcal{Q}\}$.

Input: Network $G=(\mathscr{V},\mathscr{E})$; number of type of components r; $k_i=$ number of components of type i for $i=1,\cdots,r$; sampling distribution $\{Q(\mathbf{x},\mathbf{p}),\mathbf{x}\in\mathscr{X}\}$; $\mathscr{H}^*=$ set of components types that vary in $\mathcal{Q}$; lower and upper bounds $\{g_L(\mathbf{q}),g_U(\mathbf{q});\mathbf{q}\in\mathcal{Q}\cup\{\mathbf{p}\}\}$; and number of independent replications K.

Output: $\{\tilde{g}_{0K}(\mathbf{q},\mathbf{p}),\tilde{g}_{1K}(\mathbf{q},\mathbf{p}),V[\tilde{g}_{0K}(\mathbf{q},\mathbf{p})],V[\tilde{g}_{1K}(\mathbf{q},\mathbf{p})];\mathbf{q}\in\mathcal{Q}\}$ as unbiased estimates of $\{g(\mathbf{q}),g(\mathbf{q}),\operatorname{var}\tilde{g}_{0K}(\mathbf{q},\mathbf{p}),\operatorname{var}\tilde{g}_{1K}(\mathbf{q},\mathbf{p});\mathbf{q}\in\mathcal{Q}\}$.

Method:

```
1. Initialization
   (a) $\Delta(\mathbf{p})\leftarrow g_U(\mathbf{p})-g_L(\mathbf{p})$.
   (b) For each $\mathbf{q}\in\mathcal{Q}$:
         $K(0,\mathbf{q})=K(1,\mathbf{q})\leftarrow 0$.
         For each $i\in\mathscr{H}^*$.
           $\alpha_i(\mathbf{q})\leftarrow\log[q_i(1-p_i)/p_i(1-q_i)]$and $\beta_i(\mathbf{q})\leftarrow\log[(1-q_i)/(1-p_i)]$.
2. On each of $K$ independent trials:
   (a) Sample $X_{ij}, j=1,\cdots,k_i,\ i=1,\cdots,r$ from $\{Q(\mathbf{x},\mathbf{p})\}$.
   (b) Determine $\phi(\mathbf{X})$.
   (c) For each $i\in\mathscr{H}^*$: $X_i\leftarrow\sum_{j=1}^{k_i}X_{ij}$.
   (d) Sample $Z$ from $\mathscr{U}(0,1)$.
   (e) For $\mathbf{q}\in\mathcal{Q}$:
         $T(\mathbf{q})\leftarrow 0$.
         For each $i\in\mathscr{H}^*$: $T(\mathbf{q})\leftarrow T(\mathbf{q})+k_i\beta_i(\mathbf{q})+X_i\alpha_i(\mathbf{q})$.
         $R(\mathbf{X},\mathbf{q},\mathbf{p})\leftarrow\exp[T(\mathbf{q})]$.
         $F(\mathbf{X},\phi(\mathbf{X}),\mathbf{q},\mathbf{p})\leftarrow R(\mathbf{X},\mathbf{q},\mathbf{p})/R_{\phi(\mathbf{X})}(\mathbf{q},\mathbf{p})$.
         $\varphi_{\phi(\mathbf{X})}(\mathbf{X},Z,\mathbf{q},\mathbf{p})\leftarrow\lfloor Z+F(\mathbf{X},\phi(\mathbf{X}),\mathbf{q},\mathbf{p})\rfloor$.
         $K(\phi(\mathbf{X}),\mathbf{q})\leftarrow K(\phi(\mathbf{X}),\mathbf{q})+\varphi_{\phi(\mathbf{X})}(\mathbf{X},Z,\mathbf{q},\mathbf{p})$.
```

```
3. Computation of summary statistics
    For each $\mathbf{q} \in \mathcal{Q}$:
        $\tilde{g}_{0K}(\mathbf{q}, \mathbf{p}) \leftarrow g_U(\mathbf{q}) - \Delta(\mathbf{p}) R_0(\mathbf{q}, \mathbf{p}) K(0, \mathbf{q})/K$.
        $\tilde{g}_{1K}(\mathbf{q}, \mathbf{p}) \leftarrow g_L(\mathbf{q}) + \Delta(\mathbf{p}) R_1(\mathbf{q}, \mathbf{p}) K(1, \mathbf{q})/K$.
        $V[\tilde{g}_{0K}(\mathbf{q}, \mathbf{p})] \leftarrow [\Delta(\mathbf{p}) R_0(\mathbf{q}, \mathbf{p})]^2 [K(0, \mathbf{q})/K][1 - K(0, \mathbf{q})/K]/(K-1)$.
        $V[\tilde{g}_{1K}(\mathbf{q}, \mathbf{p})] \leftarrow [\Delta(\mathbf{p}) R_1(\mathbf{q}, \mathbf{p})]^2 [K(1, \mathbf{q})/K][1 - K(1, \mathbf{q})/K]/(K-1)$.
```

In addition to computing $\{\tilde{g}_{0K}(\mathbf{q}, \mathbf{p}), \tilde{g}_{1K}(\mathbf{q}, \mathbf{p}); \mathbf{q} \in \mathcal{Q}\}$, it computes $\{V[\tilde{g}_{0K}(\mathbf{q}, \mathbf{p})], V[\tilde{g}_{1K}(\mathbf{q}, \mathbf{p})]; \mathbf{q} \in \mathcal{Q}\}$ as unbiased estimators of $\{\text{var}\, \tilde{g}_{0K}(\mathbf{q}, \mathbf{p}), \text{var}\, \tilde{g}_{1K}(\mathbf{q}, \mathbf{p}); \mathbf{q} \in \mathcal{Q}\}$. Observe that preprocessing in step 1 takes $O(|\mathcal{H}^*|w)$ time, postprocessing in step 3 takes $O(w)$ time, and, on each replication, sampling in step 2(a) takes $O(|\mathcal{E}|)$ time using Procedure Q in Fishman [5], summation in step 2(c) takes $O(\sum_{i \in \mathcal{H}^*} k_i) \leqq O(|\mathcal{E}|)$ time, and step 2(d) takes $O(|\mathcal{H}^*|w)$ time. We can also show that the mean total time for K *replications* using Algorithm A-R has the form

$$T(\{\tilde{g}_{0K}(\mathbf{q}, \mathbf{p}), \tilde{g}_{1K}(\mathbf{q}, \mathbf{p})\})$$
$$= \omega_0 + \omega_1 |\mathcal{H}^*| w + \omega_2 w + K \left[\omega_3 + \omega_4 |\mathcal{E}| + \alpha_3(\mathcal{K}_{01}, \mathbf{p})/\Delta(\mathbf{p}) + \omega_5 |\mathcal{H}^*| w + \omega_6 \sum_{i \in \mathcal{H}^*} k_i \right]$$

time where $\omega_0, \cdots, \omega_6$ denote machine-dependent constants. To reduce numerical error, all computation in step 3 should be performed in extended precision arithmetic.

Let us now compare this approach to estimating $\{g(\mathbf{q}), \mathbf{q} \in \mathcal{Q}\}$ with the alternative approach based on the w point estimates $\{\bar{g}_{K(\mathbf{q},\mathbf{p})}(\mathbf{q}), \mathbf{q} \in \mathcal{Q}\}$ using (4), where we choose the sample sizes $\{H(\mathbf{q}, \mathbf{p}), \mathbf{q} \in \mathcal{Q}\}$ to achieve equal variances under the two methods. That is,

$$\text{var}\, \bar{g}_{H(\mathbf{q},\mathbf{p})}(\mathbf{q}) = g(\mathbf{q})[1 - g(\mathbf{q})]/H(\mathbf{q}, \mathbf{p}) \tag{29}$$

where

$$H(\mathbf{q}, \mathbf{p}) = K\lambda(\mathbf{q}, \mathbf{p})$$

and

$$\lambda(\mathbf{q}, \mathbf{p}) = \frac{g(\mathbf{q})[1 - g(\mathbf{q})]}{\min_{j \in \{0,1\}} \text{var}\, \mu_j(\mathbf{X}, Z, \mathbf{q}, \mathbf{p})}.$$

Observe that

$$\lambda(\mathbf{p}, \mathbf{p}) = g(\mathbf{p})[1 - g(\mathbf{p})]/[g_U(\mathbf{p}) - g(\mathbf{p})][g(\mathbf{p}) - g_L(\mathbf{p})]$$

and, except in special cases, for any edge type $i \in \mathcal{H}^*$

$$\lim_{k_i \to \infty} \lambda(\mathbf{q}, \mathbf{p}) = 0 \quad \text{for } \mathbf{q} \neq \mathbf{p}.$$

Let

$$\lambda(\mathbf{p}) = \sum_{\mathbf{q} \in \mathcal{Q}} \lambda(\mathbf{q}, \mathbf{p}) \tag{30}$$

and observe that

$$\lim_{k_i \to \infty} \lambda(\mathbf{p}) = \lambda(\mathbf{p}, \mathbf{p}). \tag{31}$$

Therefore, the time ratio

$$\Lambda_1(\mathcal{Q}, \mathbf{p}) = \frac{T(\{\bar{g}_{H(\mathbf{q},\mathbf{p})}(\mathbf{q})\})}{T(\{\tilde{g}_{0K}(\mathbf{q}, \mathbf{p}), \tilde{g}_{1K}(\mathbf{q}, \mathbf{p})\})}, \tag{32}$$

where

$$T(\{\bar{g}_{H(\mathbf{q},\mathbf{p})}(\mathbf{q})\}) = \sum_{\mathbf{q}\in\mathcal{Q}} T(\bar{g}_{H(\mathbf{q},\mathbf{p})}(\mathbf{q})),$$

measures the efficiency of the proposed method relative to using crude Monte Carlo sampling with (4) w times to obtain estimates with equal variances var $\bar{g}_{H(\mathbf{q},\mathbf{p})}(\mathbf{q}) = \min_{j\in\{0,1\}}$ var $\tilde{g}_{jK}(\mathbf{q},\mathbf{p})$ for each $\mathbf{q}\in\mathcal{Q}$. As k_i increases, (32) assumes the form

$$(33)\qquad \begin{aligned} \Lambda_1(\mathcal{Q},\mathbf{p}) &\approx \frac{\sum_{\mathbf{q}\in\mathcal{Q}}[\alpha_2+\alpha_3(\mathcal{X},\mathbf{q})/k_i]\lambda(\mathbf{q},\mathbf{p})}{\omega_4+\alpha_3(\mathcal{X}_{01},\mathbf{p})/\Delta(\mathbf{p})k_i+\omega_6} \\ &\geqq \frac{\alpha_2+\alpha_3(\mathcal{X},\mathbf{p})/k_i}{\omega_4+\omega_6+\alpha_3(\mathcal{X}_{01},\mathbf{p})/\Delta(\mathbf{p})k_i}\lambda(\mathbf{p},\mathbf{p}) \end{aligned}$$

where the lower bound is analogous to (10). This implies that we should expect efficiency to exceed that which is obtained from estimating $g(\mathbf{p})$ only. As the example in § 8 shows, the realized efficiency can be considerably greater.

6. Individual confidence intervals. Since

$$\lim_{K\to\infty} \text{pr}\left\{\frac{|\tilde{g}_{iK}(\mathbf{q},\mathbf{p})-g(\mathbf{q})|}{[\text{var } \tilde{g}_{iK}(\mathbf{q},\mathbf{p})]^{1/2}} \leqq \beta\right\} = 2\Phi(\beta)-1$$

where $\Phi(\cdot)$ denotes the distribution function of the standard normal distribution, we can immediately compute an *approximating* confidence interval for $g(\mathbf{q})$. In particular, based on $\tilde{g} \equiv \tilde{g}_{0K}(\mathbf{q},\mathbf{p})$ and Theorem 1, we have the approximating $100\times(1-\delta)$ percent confidence interval

$$(34)\qquad \frac{\tilde{g}+[2g_U(\mathbf{q})-\Delta(\mathbf{p})R_0]\beta^2/2K \pm \beta\{\Delta(\mathbf{p})R_0\beta^2/K^2 + [g_U(\mathbf{q})-\tilde{g}][\Delta(\mathbf{p})R_0-g_U(\mathbf{q})+\tilde{g}]/K\}^{1/2}}{1+\beta^2/K}$$

for $g(\mathbf{q})$ where

$$\beta \equiv \left(z: \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{z} e^{-y^2/2}\,dy = 1-\frac{\delta}{2}\right).$$

An analogous interval can be computed based on $\tilde{g}_{1K}(\mathbf{q},\mathbf{p})$.

Because of the nonuniform convergence to normality, this approach inevitably incurs an error of approximation. An alternative approach avoids this error, albeit at the cost of a wider interval.

THEOREM 5. *Let*

$$m(z,\omega) = z\log(\omega/z)+(1-z)\log[(1-\omega)/(1-z)], \qquad 0<z, \quad \omega<1,$$

let $\omega(z,\delta/2,K)$ *denote the solution to* $m(z,\omega)=1/K\log(\delta/2)$ *for fixed* $z\in(0,1]$ *and* $\delta\in(0,1)$, *and let*

$$(35)\qquad \begin{aligned} \omega^*(z,\delta/2,K) &= \omega(z,\delta/2,K) \quad \textit{if } 0<z\leqq 1 \\ &= 0 \quad \textit{otherwise.} \end{aligned}$$

Then, the interval

(36a) (i) $(g_U(\mathbf{q})-\Delta(\mathbf{p})R_0\omega^*(1-K(0,\mathbf{q})/K,\delta/2,K),\ g_U(\mathbf{q}) - \Delta(\mathbf{p})R_0\omega^*(K(0,\mathbf{q})/K,\delta/2,K))$

covers $g(\mathbf{q})$ *with probability* $>1-\delta$, *and*

(36b) (ii) $(g_L(\mathbf{q})+\Delta(\mathbf{p})R_1\omega^*(K(1,\mathbf{q})/K, \delta/2, K), g_L(\mathbf{q})$
$+\Delta(\mathbf{p})R_1\omega^*(1-K(1,\mathbf{q})/K, \delta/2, K))$

covers $g(\mathbf{q})$ *with probability* $>1-\delta$.

The proof exploits the observation that

$$\text{pr}\,[g_U(\mathbf{q})-\Delta(\mathbf{p})R_0 \leqq \mu_0(\mathbf{X}, Z, \mathbf{q}, \mathbf{p}) \leqq g_U(\mathbf{q})]=1$$

and

$$\text{pr}\,[g_L(\mathbf{q}) \leqq \mu_1(\mathbf{X}, Z, \mathbf{q}, \mathbf{p}) \leqq g_L(\mathbf{q})+\Delta(\mathbf{p})R_1]=1.$$

The resulting confidence intervals follow from Theorem 1 of Fishman [7]. □

Since the slowest convergence to normality for $\tilde{g}_{iK}(\mathbf{q},\mathbf{p})$ occurs for $g(\mathbf{q})$ close to zero and unity, and since we are often interested in $g(\mathbf{q})$ near unity, the wider confidence intervals that result from this approach seem a reasonable price to pay to be free of the error of approximation inherent in normal intervals. Since $\{m(z,\omega)\}$ is concave in ω, we can compute the required roots by bisection.

7. Simultaneous confidence intervals. Although each confidence interval in § 6 holds with probability $>1-\delta$, the joint confidence intervals for $\{g(\mathbf{q}), \mathbf{q}\in\mathcal{Q}\}$ hold simultaneously only with probability $>1-w\delta$. This result follows from a Bonferroni inequality. See Miller [10, p. 8]. To restore the joint confidence level to $1-\delta$, we replace $\delta/2$ by $\delta/2w$ in (36a) and (36b) and determine the corresponding solutions. The effect of this substitution is to increase the constant of proportionality in the interval widths from approximately $[2\log(2/\delta)]^{1/2}$ to $[2\log(2w/\delta)]^{1/2}$ (see Fishman [7]). For $\delta=.01$ and $w=20$ we have $[\log(2w/\delta)/\log(2/\delta)]^{1/2}=1.25$. For $\delta=.01$ and $w=100$, it is 1.37 and for $\delta=.01$ and $w=1000$ it is 1.52. However, if $\mathcal{Q}$ denotes a continuous region in the $|\mathscr{E}|$-dimensional hypercube $(0,1)^{|\mathscr{E}|}$, then the resulting confidence intervals have infinite widths and are therefore useless.

For the case $Q=\{\mathbf{q}_1<\cdots<\mathbf{q}_w\}$, an alternative approach derives simultaneous confidence intervals for $\{g(\mathbf{q}), \mathbf{q}\in\mathcal{Q}\}$ by exploiting the fact that $\{K(0,\mathbf{q}_j)/K; j=1,\cdots,w\}$ and $\{K(1,\mathbf{q}_j)/K; j=1,\cdots,w\}$, in step 3 of Algorithm A-R, satisfy the definition of an empirical distribution function. Since

$$K^{-1}EK(0,\mathbf{q})=\rho(0,\mathbf{q})=[g_U(\mathbf{q})-g(\mathbf{q})]/\Delta(\mathbf{p})R_0$$

and

$$K^{-1}EK(1,\mathbf{q})=\rho(1,\mathbf{q})=[g(\mathbf{q})-g_L(\mathbf{q})]/\Delta(\mathbf{p})R_1,$$

$$\text{pr}\left\{\bigcap_{j=1}^{w}[|K(0,\mathbf{q}_j)/K-\rho(0,\mathbf{q}_j)|<d_K(\delta)]\right\}\geqq 1-\delta,$$

and

$$\text{pr}\left\{\bigcap_{j=1}^{w}[|K(1,\mathbf{q}_j)/K-\rho(1,\mathbf{q}_j)|<d_K(\delta)]\right\}\geqq 1-\delta$$

where $d_K(\delta)$ denotes the critical value of the Kolmogorov-Smirnov distribution for sample size K at significance level δ. Therefore,

(37a) $\quad g_U(\mathbf{q}_j)-\Delta(\mathbf{p})R_0(\mathbf{q}_j,\mathbf{p})[K(0,\mathbf{q}_j)/K\pm d_K(\delta)] \quad \forall j=1,\cdots,w$

cover $g(\mathbf{q}_1), \cdots, g(\mathbf{q}_w)$ simultaneously with probability $\geqq 1-\delta$ and similarly

$$g_L(\mathbf{q}_j)+\Delta(\mathbf{p})R_1(\mathbf{q}_j,\mathbf{p})[K(1,\mathbf{q}_j)/K \mp d_K(\delta)] \quad \forall j=1,\cdots,w \tag{37b}$$

cover $q(\mathbf{q}_1), \cdots, g(\mathbf{q}_w)$ with probability $\geqq 1-\delta$. For $\delta=.05$, $\lim_{K\to\infty} K^{1/2}d_K(.05)=1.3581$ and for $\delta=.01$ $\lim_{K\to\infty} K^{1/2}d_K(.01)=1.6276$. Since $d_\infty(.05)/d_K(.05)\leqq 1.013$ for $K\geqq 100$ and $d_\infty(.01)/d_K(.01)\leqq 1.014$ for $K\geqq 80$ (Birnbaum [4]), little error arises when replacing $d_K(.05)$ by $1.3581/K^{1/2}$ and $d_K(.01)$ by $1.6276/K^{1/2}$ above for $K\geqq 100$. The appeal of this alternative approach is that the widths of the intervals are all independent of w. The limitation is that all intervals are of the same width. In practice, we can compute the intervals based on (36a) and (36b) with $\delta/2w$ replacing $\delta/2$ and the intervals based on (37a) and (37b), and choose the set with smaller widths.

8. Example. An analysis of the network in Fig. 1 illustrates the proposed method. The network has 30 edges and 20 nodes. The example assumes $r=1$ so that all edges have identical reliabilities, allowing us to write $\mathbf{q}=q$. Note that any other specification with $r>1$ can be accommodated easily. The objective is to estimate $\{g(\mathbf{q}), q=.80+.01(i-1), i=1,\cdots,20\}$ where $g(\mathbf{q})=$ probability that nodes $s=1$ and $t=20$ are connected when edge reliabilities are q. For sampling, we use $\mathbf{p}=p$, again merely as a convenience. The lower and upper bounding functions $\{g_L(\mathbf{q})\}$ and $\{g_U(\mathbf{q})\}$ were computed beforehand using edge-disjoint minimal s-t cutsets for $\{g_L(\mathbf{q})\}$ and edge-disjoint minimal s-t cutsets for $\{g_U(\mathbf{q})\}$, as in Fishman [5]. To determine these paths takes $O(I|\mathscr{E}|)$ time, where I denotes the size of the smallest minimal s-t cutset and to determine the cutsets takes $O(|\mathscr{E}|)$ time. The determination of R_0 and R_1 is discussed in Fishman [7]. The evaluation of $\phi(\mathbf{X})$ using a depth-first search as in Aho, Hopcroft, and Ullman [1] takes $O(\max(|\mathscr{E}|,|\mathscr{V}|))$ time.

An experiment was run with $p=.80$, which minimized the worst-case variances as in (24), and with sample size $K=2^{20}=1{,}048{,}576$. Since results for $\{\tilde{g}_{0K}(\mathbf{q},\mathbf{p})\}$ were considerably more favorable than those for $\{\tilde{g}_{1K}(\mathbf{q},\mathbf{p})\}$, the analysis focuses on $\{\tilde{g}_{0K}(\mathbf{q},\mathbf{p})\}$. Table 1 shows individual point estimates and confidence intervals, the latter having been computed as in (36a). Table 2 compares the precomputed worst case and the empirically observed coefficients of variation and variances, and Table 3

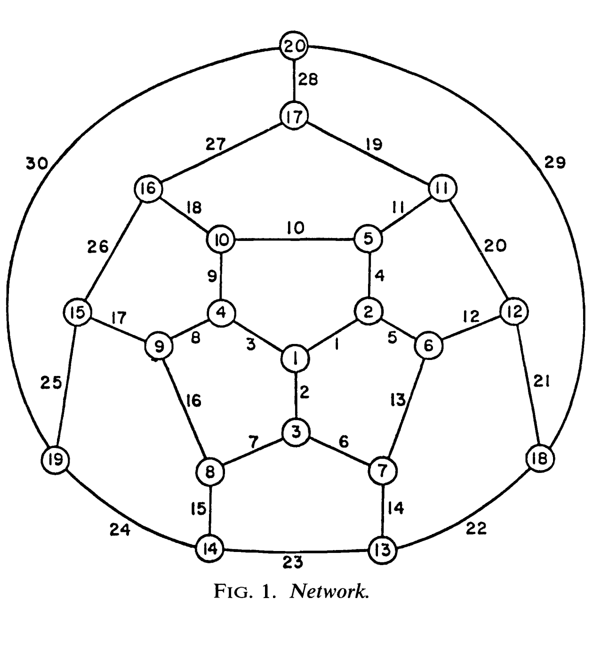


FIG. 1. *Network.*

TABLE 1
Reliability estimation.
$(p=.80,\ K=1{,}048{,}576)$

						Individual 99% confidence intervals on $1-g(q)$		
q	$1-g_U(q)$ (1)	$1-g_L(q)$ (2)	$1-g(q)$† (3)	$1-\tilde{g}_{0K}(q,p)$‡ (4)	$V[\tilde{g}_{0K}(q,p)]$‡ (5)	Lower (6)	Upper (7)	Width (8)
.80	.1612D−01	.3039D+00	.3624D−01	.3628D−01	.5144D−08	.3604D−01	.3651D−01	.4669D−03
.81	.1381D−01	.2763D+00	.2976D−01	.2976D−01	.4683D−08	.2954D−01	.2999D−01	.4455D−03
.82	.1173D−01	.2492D+00	.2421D−01	.2420D−01	.4120D−08	.2400D−01	.2441D−01	.4179D−03
.83	.9874D−02	.2227D+00	.1949D−01	.1953D−01	.3512D−08	.1933D−01	.1972D−01	.3858D−03
.84	.8225D−02	.1969D+00	.1552D−01	.1559D−01	.2893D−08	.1542D−01	.1577D−01	.3502D−03
.85	.6773D−02	.1722D+00	.1221D−01	.1227D−01	.2282D−08	.1212D−01	.1243D−01	.3110D−03
.86	.5503D−02	.1485D+00	.9473D−02	.9540D−02	.1728D−08	.9406D−02	.9677D−02	.2707D−03
.87	.4404D−02	.1262D+00	.7241D−02	.7240D−02	.1223D−08	.7127D−02	.7355D−02	.2276D−03
.88	.3462D−02	.1053D+00	.5440D−02	.5442D−02	.8354D−09	.5349D−02	.5537D−02	.1882D−03
.89	.2666D−02	.8611D−01	.4006D−02	.4017D−02	.5407D−09	.3942D−02	.4094D−02	.1514D−03
.90	.2002D−02	.6867D−01	.2880D−02	.2876D−02	.3197D−09	.2819D−02	.2935D−02	.1164D−03
.91	.1459D−02	.5314D−01	.2011D−02	.2002D−02	.1739D−09	.1960D−02	.2046D−02	.8587D−04
.92	.1025D−02	.3962D−01	.1355D−02	.1354D−02	.8754D−10	.1325D−02	.1385D−02	.6093D−04
.93	.6862D−03	.2818D−01	.8720D−03	.8820D−03	.4039D−10	.8620D−03	.9034D−03	.4139D−04
.94	.4321D−03	.1884D−01	.5284D−03	.5334D−03	.1490D−10	.5213D−03	.5465D−03	.2514D−04
.95	.2500D−03	.1158D−01	.2947D−03	.2984D−03	.4515D−11	.2918D−03	.3056D−03	.1384D−04
.96	.1280D−03	.6293D−01	.1456D−03	.1467D−03	.9371D−12	.1437D−03	.1500D−03	.6307D−05
.97	.5400D−04	.2819D−01	.5937D−04	.5981D−04	.1209D−12	.5875D−04	.6102D−04	.2267D−05
.98	.1600D−04	.8869D−03	.1702D−04	.1710D−04	.5882D−14	.1686D−04	.1736D−04	.5000D−06
.99	.2000D−05	.1177D−03	.2062D−05	.2067D−05	.2941D−16	.2051D−05	.2087D−05	.3537D−07

† Provided by J. S. Provan using an algorithm based on cutset enumeration.
‡ Computed as in Algorithm A-R.

TABLE 2
Coefficients of variation and variances.
($p = .80$)

	$\gamma_0(q,p)=\frac{[\operatorname{var}\mu_0(\mathbf{X},Z,q,p)]^{1/2}}{1-E\mu_0(\mathbf{X},Z,q,p)}$		$\operatorname{var}\mu_0(\mathbf{X},Z,q,p)$	
q	Worst case[1]	Observed[2]	Worst case[3]	Observed[2]
.80	2.06	2.02	.207D−01	.539D−02
.81	2.37	2.35	.262D−01	.491D−02
.82	2.72	2.72	.322D−01	.432D−02
.83	3.11	3.11	.383D−01	.368D−02
.84	3.53	3.53	.435D−01	.303D−02
.85	4.00	3.93	.455D−01	.239D−02
.86	4.51	4.46	.443D−01	.181D−02
.87	5.06	4.95	.406D−01	.128D−02
.88	5.64	5.44	.349D−01	.876D−03
.89	6.26	5.93	.282D−01	.567D−03
.90	6.91	6.37	.212D−01	.335D−03
.91	7.57	6.74	.147D−01	.182D−03
.92	8.23	7.08	.928D−02	.918D−04
.93	8.87	7.37	.520D−02	.423D−04
.94	9.44	7.41	.250D−02	.156D−04
.95	9.89	7.29	.982D−03	.473D−05
.96	10.13	6.76	.287D−03	.983D−06
.97	10.04	5.95	.527D−04	.127D−06
.98	9.37	4.59	.415D−05	.617D−08
.99	7.55	2.69	.396D−07	.308D−10

[1] Computed from (21).
[2] Estimated from data.
[3] Computed from (20).

shows the worst case and empirically observed variance ratios, where the variance in the numerator corresponds to that for crude Monte Carlo sampling.

Recall that the worst-case results can be computed and used prior to sampling. For example, suppose that we want a coefficient of variation no larger than $u_* = .01$ for all point estimates. Since the largest worst-case result in Table 2 is 10.13, we would use (25) to compute the worst-case sample size $\eta_{**} = 1{,}008{,}016$. All results in columns 4–8 of Table 1 took 74.9 minutes to compute in total, or 4.28 milliseconds per replication.

This network was chosen for illustration because its computing time fit within the budget available for computing. As intended, it clearly demonstrates the superiority of the proposed technique when compared to crude Monte Carlo sampling. Shier [12] and Page and Perry [11] provide exact solutions to this particular problem with evaluation times considerably smaller than our experiment takes, and an analyst contemplating the evaluation of reliability for networks of this size is well advised to consider these exact methods. However, as the size of the network under consideration grows, these exact methods inevitably show their exponential time growth, whereas the time per replication for the Monte Carlo approach remains $O(\max(|\mathscr{V}|, |\mathscr{E}|))$. Therefore, for substantially larger networks there will always exist sets of component reliabilities $\mathscr{Q}$ such that estimating $\{g(\mathbf{q}), \mathbf{q} \in \mathscr{Q}\}$ as proposed here will be more efficient timewise than these exact methods of solution.

9. A comparison. At least one alternative method exists for using the data from a single experiment with input vector $\mathbf{p}$ to generate estimates of $\{g(\mathbf{q}), \mathbf{q} \in \mathscr{Q}\}$. This

TABLE 3
Variance ratios.

$$\frac{\operatorname{var} \bar{g}_K(q)}{\operatorname{var} \tilde{g}_{0K}(q, p)}$$

q	Worst case[1]	Observed[2]
.80	5.15	6.48
.81	4.32	5.88
.82	3.71	5.47
.83	3.27	5.20
.84	2.95	5.06
.85	2.75	5.07
.86	2.60	5.21
.87	2.54	5.61
.88	2.56	6.18
.89	2.68	7.06
.90	2.91	8.55
.91	3.31	10.95
.92	3.97	14.73
.93	5.10	20.81
.94	7.14	34.12
.95	11.23	63.00
.96	20.88	149.23
.97	50.53	471.61
.98	197.50	2771.75
.99	2490.13	67040.00

[1] Computed from (27).
[2] Estimated from data.

method is based on using the *importance function* (13) to form

$$\psi_a(\mathbf{x}, \mathbf{q}, \mathbf{p}) = g_L(\mathbf{q}) + \Delta(\mathbf{p})\phi(\mathbf{x})R(\mathbf{x}, \mathbf{q}, \mathbf{p})$$

and

$$\psi_b(\mathbf{x}, \mathbf{q}, \mathbf{p}) = g_U(\mathbf{q}) + \Delta(\mathbf{p})[1 - \phi(\mathbf{x})]R(\mathbf{x}, \mathbf{q}, \mathbf{p})$$

so that $\psi_a(\mathbf{X}, \mathbf{q}, \mathbf{p})$ and $\psi_b(\mathbf{X}, \mathbf{q}, \mathbf{p})$ both have expectation $g(\mathbf{q})$ when $\mathbf{X}$ is from $\{Q(\mathbf{x}, \mathbf{p})\}$. Fishman [6] studies these estimates in detail using the same network, and a comparison between these importance functions (IF) and the currently proposed acceptance-rejection (A-R) estimators seems appropriate.

For every $\mathbf{q} \in \mathcal{Q}$, the IF estimators have smaller variance that the A-R estimators do and both methods have about the same computation time per replication. If variance is the dominant consideration, then the IF method prevails. However, there are other issues that also deserve consideration. The A-R estimators have considerably simpler expressions for variance and coefficient of variation than the IF estimators do. Also, on each trial $\mu_0(\mathbf{X}, Z, \mathbf{q}, \mathbf{p})$ and $\mu_1(\mathbf{X}, Z, \mathbf{q}, \mathbf{p})$ for the A-R approach each assume binary values thus allowing standard techniques of analysis for binary data to apply. In contrast $\psi_a(\mathbf{X}, \mathbf{q}, \mathbf{p})$ and $\psi_b(\mathbf{X}, \mathbf{q}, \mathbf{p})$ in the IF approach each assume $O(\prod_{i=1}^{r} (k_i + 1))$ values precluding the use of the simpler analysis.

With regard to confidence intervals, the A-R approach allows us to compute individual asymptotically normal intervals without nuisance parameters, whereas the IF estimators do not. For individual confidence intervals based on Theorem 5, both methods give intervals of about the same length. This is a consequence of ignoring estimated variance information for the IF method. For simultaneous confidence intervals the A–R method allows the development in § 7 when $\mathbf{q}_1, \cdots, \mathbf{q}_w$, are ordered, whereas the IF method does not.

The IF method relies on the quantities $R_0(\mathbf{q}, \mathbf{p})$ and $R_1(\mathbf{q}, \mathbf{p})$ to derive confidence intervals whereas the A–R method uses them to determine acceptance or rejection. If these quantities are difficult to compute, then we may derive upper bounds for them, as in Fishman [6], resulting in wider confidence intervals for the IF method and lower acceptance frequencies for the A–R method.

Acknowledgment. I am grateful to Christos Alexopoulos for his many helpful comments on this manuscript and for his programming assistance.

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] R. E. BARLOW AND F. PROSCHAN, *Statistical theory of reliability and life testing probability models*, To Begin With, Silver Spring, MD, 1981.

[3] R. J. BECKMAN AND M. D. MCKAY, *Monte Carlo estimation under different distributions using the same simulation*, Technometrics, 29 (1987), pp. 153-160.

[4] Z. W. BIRNBAUM, *Numerical tabulation of the distribution of Kolmogorov's statistic for finite sample size*, J. Amer. Statist. Assoc., 47 (1952), pp. 425-441.

[5] G. S. FISHMAN, *A Monte Carlo sampling plan for estimating network reliability*, Oper. Res., 34 (1986), pp. 581-594.

[6] ———, *Sensitivity analysis for the system reliability function*, Tech. Report UNC/OR-87/6, Department of Operations Research, University of North Carolina, Chapel Hill, NC, 1987; submitted for publication.

[7] ———, *Confidence intervals for mean and proportions in the bounded case*, Tech. Report UNC/OR/TR-86/19, Department of Operations Research, University of North Carolina, Chapel Hill, NC, revised 1989; submitted for publication.

[8] H. KAHN, *Random sampling (Monte Carlo) techniques in neutron attenuation problems*—I, Nucleonics, 6 (1950), pp. 27-33.

[9] H. KAHN AND T. E. HARRIS, *Estimation of particle transmissions by random sampling*, in Monte Carlo Methods, National Bureau of Standards, Applied Mathematics Series, 12, Washington, DC, 1951.

[10] R. MILLER, *Simultaneous Statistical Inference*, 2nd ed., Springer-Verlag, Berlin, New York, 1981.

[11] L. B. PAGE AND J. E. PERRY, *A practical implementation of the factoring theorem for network reliability*, IEEE Trans. Reliablity, 37 (1988), pp. 259-267.

[12] D. SHIER, *Algebraic aspects of computing network reliability*, in 3rd Conference on Discrete Mathematics; R. Ringeisen and F. Roberts, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988, pp. 135-147.

[13] J. VON NEUMANN, *Various techniques used in connection with random digits*, in Monte Carlo Method, A. S. Householder, G. E. Forsythe, and H. H. Germond, eds., National Bureau of Standards Applied Mathematics Series 12, Washington, DC, issued June 11, 1951.

SIAM J. Sci. Stat. Comput.
Vol. 11, No. 6, pp. 1181–1192, November 1990

A NEW ALGORITHM FOR NUMERICAL PATH FOLLOWING APPLIED TO AN EXAMPLE FROM HYDRODYNAMICAL FLOW*

JACQUES HUITFELDT† AND AXEL RUHE†

Abstract. A numerical algorithm for following a path of solutions to a nonlinear eigenvalue problem is described. It is an Euler–Newton continuation method, where the linear systems are solved with an Arnoldi iteration, where a factorization of the Jacobian matrix at an earlier point is used as a preconditioner. It is shown how the eigenvalues of the Hessenberg matrix produced in the Arnoldi iteration can be used to localize singular points, i.e., turning points or bifurcations along the path. A case of the Taylor problem from hydrodynamics is reported as a numerical example.

Key words. bifurcation, nonlinear equations, eigenvalue, Arnoldi algorithm, spectral transformation

AMS(MOS) subject classifications. 65H15, 65H25

1. Introduction.

In the present contribution we describe a numerical algorithm for following paths of solutions and localize turning points and bifurcations of nonlinear eigenvalue problems of the form

$$F(x, \lambda) = 0, \tag{1.1}$$

where $F:\mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$. Usually λ is a real physical parameter (i.e., Reynolds number in hydrodynamical flow, load in structural mechanics, etc.) and $x = x(\lambda)$ represents an approximation to the solution (i.e., flow field, displacements, etc.).

We have focused our interest on large and sparse cases where the path following involves solution of large linear systems, but our most notable finding is a new way of predicting singular points along the way, using a spectral transformation of the kind that earlier proved to be successful for linear eigenvalue problems [7].

In § 2 we will briefly recall the Euler–Newton continuation method, which is explained in the pioneering papers of Keller [8], Rheinboldt [11], and Riks [12]. Furthermore, we describe a method, proposed in [12], for prediction of singular points along the path, by solving a linear eigenvalue problem (see also [13]).

We continue in § 3 by describing how a factorization of the Jacobian matrix in one point can be used as a preconditioner in an iterative solution of linear systems in later points. We also show how this technique can provide the information needed to predict singular points along the path. In [3] and [4] other preconditioning techniques have been proposed. We believe our preconditioning is a new approach. A preliminary version of the material of § 3 has appeared in [16]. In § 4 we describe how to overcome difficulties at singular points and how to switch branches. We have used an adaptation of methods described in [8].

In the last section we will test the new method on one example of the Taylor problem of hydrodynamics, which is the steady axisymmetric flow of an incompressible viscous fluid between two concentric rotating circular cylinders. The steady-state Navier–Stokes equations govern the flow and a discretization gives a problem of the form (1.1). This problem is interesting because it is a nontrivial example that has been tried by several other authors, see, for instance, [9] and [10].

* Received by the editors November 2, 1988; accepted for publication (in revised form) October 31, 1989.

† Department of Computer Science, Chalmers University of Technology and the University of Göteborg, S-412 96 Göteborg, Sweden.

Let A be a real $n\times n$ matrix; then we denote by $\mathbf{N}(A)$ the null space of A, $\{z\in\mathbb{R}^n\colon Az=0\}$ and by $\mathbf{R}(A)$ the range space of A, $\{w\in\mathbb{R}^n\colon w=Ay \text{ for some } y\in\mathbb{R}^n\}$. The vector norm used is the usual Euclidean vector norm, $\|x\|_2=(x^Tx)^{1/2}$. e_k is the coordinate vector with a 1 in the kth position.

2. Natural parametrization. If (x_0,λ_0) is a known solution of our problem (1.1) and the matrix of partial derivatives $F_x(x_0,\lambda_0)$ is regular, then (x_0,λ_0) is called a regular point, and the implicit function theorem guarantees the existence of a unique path of solutions $(x(\lambda),\lambda)$ through (x_0,λ_0), parametrized by the naturally occurring parameter λ, with $x(\lambda_0)=x_0$. If $F_x(x_0,\lambda_0)$ is singular then (x_0,λ_0) is called a turning point if the nullspace is one-dimensional

$$\dim \mathbf{N}(F_x(x_0,\lambda_0))=1, \tag{2.1}$$

and the λ derivative is not in the range

$$F_\lambda(x_0,\lambda_0)\notin \mathbf{R}(F_x(x_0,\lambda_0)). \tag{2.2}$$

That means that the tangent to the path becomes orthogonal to the λ direction. If, on the other hand, the λ derivative is in the range space, then (x_0,λ_0) is called a simple bifurcation point. We postpone the discussion of nullspaces of higher dimensions.

The basic tool for the numerical computation of a path of solutions is the use of a continuation method. All continuation methods depend on the implicit function theorem and are based on some predictor-corrector scheme. The continuation method that is most commonly used is the Euler–Newton method, which we now describe.

2.1. The Euler–Newton continuation method. Assume (x_0,λ_0) is a known regular solution of (1.1) and that we want to compute the solution x_1 at $\lambda=\lambda_1$. Differentiation of (1.1) with respect to λ gives the differential algebraic system

$$F_x(x(\lambda),\lambda)x'(\lambda)=-F_\lambda(x(\lambda),\lambda). \tag{2.3}$$

We predict the solution x_1 at λ_1 with an Euler step,

$$x_1^{(0)}=x_0+(\lambda_1-\lambda_0)x'(\lambda_0), \tag{2.4}$$

where $x'(\lambda_0)$ is found from (2.3). Now we use this prediction as an initial guess to solve $F(x,\lambda_1)=0$ with the Newton method, solving systems

$$F_x(x_1^{(i)},\lambda_1)(x_1^{(i+1)}-x_1^{(i)})=-F(x_1^{(i)},\lambda_1),\qquad i=0,1,\cdots, \tag{2.5}$$

until convergence.

A continuation method may fail or encounter difficulties when approaching a singular point along the path. At a turning point these problems can be overcome by augmenting equation (1.1) with an artificial continuation parameter and some additional constraint or normalization. At a bifurcation point some method for switching from one branch to another is needed. In both cases it is desirable to be able to predict the location of the singularity.

2.2. Prediction of singular points along the path. Suppose we have a solution path $(x(\lambda),\lambda)$. For what value of λ is $A(\lambda)=F_x(x(\lambda),\lambda)$ singular? Instead of solving the usually nonlinear eigenvalue problem $A(\lambda)u=0$, we make a linearization

$$A(\lambda)=A(\lambda_0)+(\lambda-\lambda_0)A'(\lambda_0)+\tfrac{1}{2}(\lambda-\lambda_0)^2R(\lambda), \tag{2.6}$$

where $A'(\lambda)=F_{xx}(x(\lambda),\lambda)x'(\lambda)+F_{x\lambda}(x(\lambda),\lambda)$ and $R(\lambda)$ is bounded if F is sufficiently differentiable, and get a generalized linear eigenvalue problem,

$$A(\lambda_0)w+\nu A'(\lambda_0)w=0. \tag{2.7}$$

Now if ν_0 and w_0 solve this eigenvalue problem, then $\lambda_s = \lambda_0 + \nu_0$ gives a prediction of λ at a singular point. Furthermore, if ν_0 is close to zero we have a singular point close to λ_0. Then $(w_0, 0)$ gives an approximation to the tangent to the path at the singular point, if it is a turning point. If it is a simple bifurcation point, $(x'(\lambda_0), \lambda_0)$ and $(w_0, 0)$ span an approximation to the tangent plane at the bifurcation point, provided that the norm $\|x'(\lambda_0)\|_2$ is small enough. In § 4 we will augment equation (1.1) with an additional equation. The corresponding augmented eigenvalue problem will always give an approximation to the tangent line or tangent plane at the singular point, if some eigenvalue is close to zero.

3. The preconditioned Arnoldi iteration. Since factorization of large matrices is costly, solving linear systems by a direct method in each continuation step is unfavorable for large problems. Different iterative methods have been proposed (see, for instance, [2] and the references cited therein). The number of iterations is related to the condition number of the matrix, and in ill-conditioned cases it is reduced if an appropriate preconditioning technique is used. We have chosen to factor F_x in some selected points, and use these factors as a preconditioner in the Arnoldi iteration in a sequence of continuation steps. When many Arnoldi iterations are needed, we select a new factorization point.

In § 3.1 we recall the preconditioned Arnoldi algorithm and in § 3.2 we describe how to approximate the solutions of the eigenvalue problem (2.7) in terms of quantities available during the computation. We summarize the computations in an algorithm in the last section.

3.1. Solution of linear systems. Suppose we know a solution point (x_0, λ_0) and that we have factored the Jacobian $M \equiv F_x(x_0, \lambda_0)$ there. In each continuation step we want to solve several linear systems of the form $F_x d = b$ (see (2.3) and (2.5)) where the Jacobian F_x is evaluated near or on the solution path. Multiplication with M^{-1} from the left gives the system

$$M^{-1} F_x d = M^{-1} b. \tag{3.1}$$

If M is close to F_x then $M^{-1}F_x$ is close to the identity matrix, and we can expect fast convergence in the Arnoldi iteration. Given an approximation d_0 to the solution, the algorithm can be described as follows.

ALGORITHM PA (preconditioned Arnoldi).

```
1. Compute $r_0 := M^{-1}(b - F_x d_0)$, and take $v_1 := r_0/\beta$, where $\beta := \|r_0\|_2$.
2. For $j = 1, 2, \cdots$, until convergence.
   (1) $r := M^{-1} F_x v_j$
   (2) For $i = 1, 2, \cdots, j$
           $h_{ij} := v_i^T r$
           $r := r - v_i h_{ij}$
   (3) $h_{j+1j} := \|r\|_2$
       $v_{j+1} := r / h_{j+1j}$
   (4) Test for convergence.
```

After m steps we have

$$M^{-1} F_x V_m - V_m H_{mm} = h_{m+1m} v_{m+1} e_m^T, \tag{3.2}$$

where H_{mm} is an upper Hessenberg matrix with the h_{ij}'s as its nonzero entries and V_m is an orthogonal matrix with the v_j's as columns. V_m is a basis of the Krylov subspace

$$K^m_{M^{-1}F_x}(r_0) = \text{span}\,\{r_0, (M^{-1}F_x)r_0, \cdots, (M^{-1}F_x)^{m-1} r_0\}. \tag{3.3}$$

If we solve the (small) linear system,

$$H_{mm}y_m = \beta e_1, \tag{3.4}$$

we obtain a new approximate solution

$$d_m = d_0 + V_m y_m \tag{3.5}$$

to $F_x d = b$. The residual norm can be estimated by the formula

$$\|r_m\|_2 = \|M^{-1}(b - F_x d_m)\|_2 = h_{m+1m}|e_m^T y_m|, \tag{3.6}$$

which follows immediately from the relation (3.2). Note that this estimate can be computed without actually forming d_m, only the system (3.4) must be solved. If a subdiagonal element $h_{m+1m} = 0$, then d_m is the exact solution of the linear system. In practice this seldom happens, but we can hope that eventually the last component of the vector y_m will become very small.

In [1] a local convergence theory is presented for Newton's method in connection with Krylov subspace methods, and in [14] Krylov subspace methods for solving linear systems are described. We have used the solution named FOM in [6]. We have not tested the GMRES solution [15], but believe that the distinction is very small in this case.

3.2. Prediction of singular points by Arnoldi. Let us now turn to the problem of predicting singular points along the path. When we use the Algorithm PA of the preceding section, we can avoid the problem of finding the derivative (2.7), if we replace the Taylor expansion (2.6) by the Lagrange interpolation formula

$$A(\lambda) = \frac{\lambda - \lambda_1}{\lambda_0 - \lambda_1} A(\lambda_0) + \frac{\lambda - \lambda_0}{\lambda_1 - \lambda_0} A(\lambda_1) + \frac{1}{2}(\lambda - \lambda_0)(\lambda - \lambda_1) R(\lambda), \tag{3.7}$$

where $R(\lambda)$ is bounded if F is sufficiently differentiable.

An approximative solution to $A(\lambda)u = 0$ is then given by solving

$$[(\lambda - \lambda_1)A(\lambda_0) - (\lambda - \lambda_0)A(\lambda_1)]w = 0,$$

or

$$A(\lambda_0)^{-1}A(\lambda_1)w = w\theta \tag{3.8}$$

where

$$\theta = \frac{\lambda - \lambda_1}{\lambda - \lambda_0}, \tag{3.9}$$

or

$$\lambda = \lambda_1 + \frac{\theta}{1 - \theta}(\lambda_1 - \lambda_0).$$

The interesting fact is now that, if we let λ_0 be the factorization point in the preconditioned Arnoldi algorithm and λ_1 be the point where the solution is sought, then in formula (3.2)

$$M = A(\lambda_0), \qquad F_x = A(\lambda_1),$$

and the Arnoldi iteration will yield approximations to the transformed increment θ (3.9), as eigenvalues of the Hessenberg matrix

$$H_{mm}s_m = s_m\theta_m, \qquad w_m = V_m s_m. \tag{3.10}$$

We note from the spectral transformation (3.9) plotted in Fig. 3.1, that the singular points λ closest to the $[\lambda_0, \lambda_1]$ interval correspond to the absolutely largest eigenvalues θ of (3.8). These are precisely those that will converge first in the Arnoldi process (3.10), giving us a good prediction of the closest singular points already at a very small number of iteration steps in Algorithm PA. Usually the θ are in the interval $0<\theta<2$. A negative θ means that there is a singular point between λ_0 and λ_1, and a $\theta \geqq 2$ means that there is a singularity close behind λ_0. In this paper we are only interested in real λ-values.

3.3. The preconditioned Arnoldi continuation. Let us now summarize the algorithm we have described. Given a regular solution (x_0, λ_0) of (1.1), the following algorithm will follow a path of solutions $(x(\lambda), \lambda)$ and predict the location of singularities along the path.

```
ALGORITHM PAC (preconditioned Arnoldi continuation).
1. Make a factorization of the Jacobian $M \equiv F_x(x_0, \lambda_0)$.
2. For $k = 0, 1, \cdots$
    (1) Compute $x'(\lambda_k)$ from (2.3) using Algorithm PA with $F_x = F_x(x_k, \lambda_k)$ and
        $b = -F_\lambda(x_k, \lambda_k)$.
    (2) If $k \neq 0$ then: Solve (3.10), with $H_{mm}$ from step (1), to predict the location
        of the singularities, $\lambda_s = \lambda_k + \theta_m(1-\theta_m)^{-1}(\lambda_k - \lambda_0)$.
    (3) Choose $\lambda_{k+1}$ and perform the Euler step (2.4), to predict the solution at $\lambda_{k+1}$.
    (4) Perform the Newton correction (2.5) to obtain the solution $x_{k+1}$ at $\lambda_{k+1}$.
        Solve the linear systems using Algorithm PA with $F_x = F_x(x_{k+1}^{(i)}, \lambda_{k+1})$ and
        $b = -F(x_{k+1}^{(i)}, \lambda_{k+1})$.
```

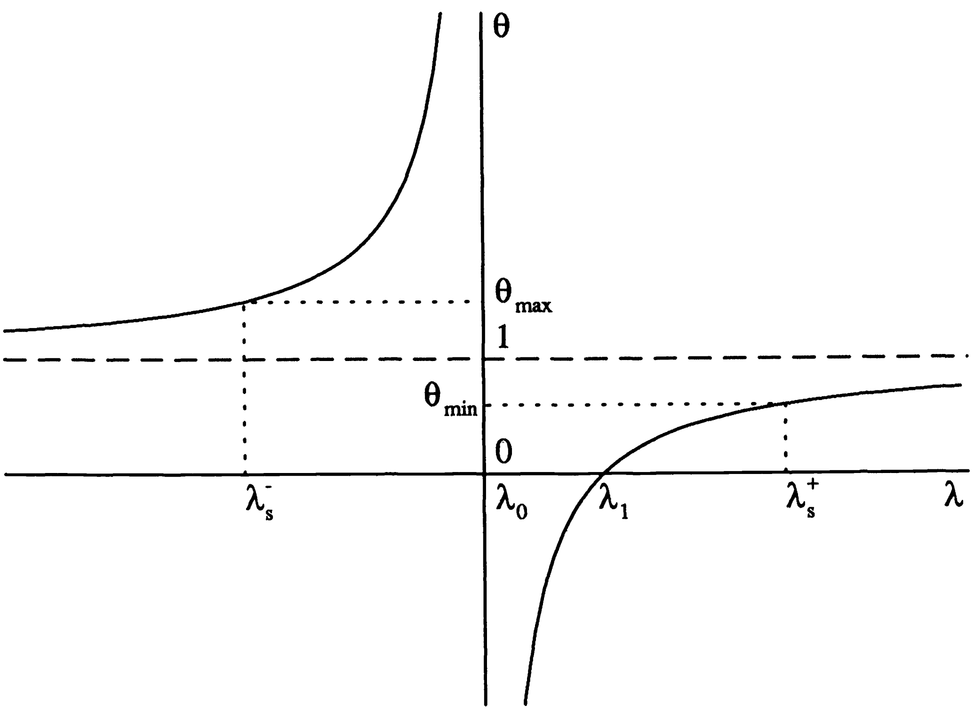


FIG. 3.1. *The spectral transformation. Each eigenvalue θ of* (3.8) *corresponds to a predicted singular λ-value, $\lambda_s = \lambda_1 + \theta(1-\theta)^{-1}(\lambda_1 - \lambda_0)$. Note that a θ-value less* (*greater*) *than one corresponds to a singularity with λ-value greater* (*less*) *than λ_0.*

(5) If too many Arnoldi iterations are needed to solve the linear systems, use (x_{k+1}, λ_{k+1}) as the new (x_0, λ_0) and return to step 1.

Let us now comment on step 2(2).

We can estimate the residual norm of the eigenvalue problem (3.8) by the formula

$$\|A(\lambda_0)^{-1}A(\lambda_k)w_m - w_m\theta_m\|_2 = h_{m+1m}|e_m^T s_m|, \tag{3.11}$$

which follows immediately from the relation (3.2). This estimate can be computed without forming the large vector w_m, only the small eigenvalue problem (3.10) must be solved. If this estimate is not small enough, for a real positive θ_m, either we restart Algorithm PA with a random starting vector, if m is very small, or we continue Algorithm PA with the same vector, until the residual norm is small enough. In the latter case we reorthogonalize the basis matrix V_m.

We address some more implementation issues in § 5.2.

4. Augmented problem. We can follow the solution path beyond turning points in λ, if we augment equation (1.1) with an additional artificial parameter, which is often related to arclength and some constraint or normalization. Consider the equation

$$H(y, \sigma) \equiv \begin{bmatrix} F(x, \lambda) \\ N(x, \lambda, \sigma) \end{bmatrix} = 0, \tag{4.1}$$

where $H: \mathbb{R}^{n+1} \times \mathbb{R} \to \mathbb{R}^{n+1}$ and $y \equiv (x, \lambda)$. We can apply the implicit function theorem to this new equation (4.1), provided that the Jacobian, with respect to y,

$$H_y(y, \sigma) \equiv \begin{bmatrix} F_x(x, \lambda) & F_\lambda(x, \lambda) \\ N_x(x, \lambda, \sigma) & N_\lambda(x, \lambda, \sigma) \end{bmatrix} \tag{4.2}$$

is regular. If the dimension of the nullspace of F_x is at most one, it is easy to show that at a regular point of (1.1), H_y is regular if and only if

$$N_\lambda - N_x F_x^{-1} F_\lambda \neq 0 \tag{4.3}$$

and at a singular point of (1.1), H_y is regular if and only if

$$F_\lambda \notin \mathbf{R}(F_x), \tag{4.4}$$

$$N_x \notin \mathbf{N}(F_x)^\perp. \tag{4.5}$$

Condition (4.3) simply states that the normal vector to the surface defined by the equation $N(x, \lambda, \sigma) = 0$, may not be orthogonal to the tangent to the solution path. If (4.4) holds we can always choose N in (4.1) such that (4.3) and (4.5) hold. We note that condition (4.4) holds at turning points of (1.1) but not at bifurcation points.

For properly chosen N, regular points and turning points of (1.1) are regular points of (4.1). Hence, provided an initial solution (x_0, λ_0) is known, a continuation method applied to the augmented equation (4.1) will compute connected components of the solution manifold of (1.1) containing regular points and turning points.

4.1. Additional equation. Several different choices of additional constraints have been proposed. Some often used constraints are

$$N(x, \lambda, \sigma) \equiv \lambda - \sigma, \tag{4.6}$$

$$N(x, \lambda, \sigma) \equiv x_k - \sigma, \tag{4.7}$$

$$N(x, \lambda, \sigma) \equiv \|x'(\sigma)\|^2 + |\lambda'(\sigma)|^2 - 1, \tag{4.8}$$

$$N(x, \lambda, \sigma) \equiv x'(\sigma_0)^T(x(\sigma) - x_0) + \lambda'(\sigma_0)(\lambda(\sigma) - \lambda_0) - (\sigma - \sigma_0). \tag{4.9}$$

Equation (4.6) gives the natural parametrization, a coordinate direction (4.7) for a proper choice of k is used by Rheinboldt and others (see [11]), the arclength gives the constraint (4.8), and the normalization (4.9) is the pseudo arclength used by Keller [8] and Riks [12].

We choose to use the tangent direction, at the point where we factor the matrix, as parameter direction for several steps. This corresponds to a change of coordinates. When we make a new factorization or approach a singular point, we take a new tangent direction. That is, we use the same pseudo arclength equation (4.9) for several steps in the continuation method in order to be able to make consistent predictions of the location of singularities.

4.2. Prediction of singular points. We can follow a path of solutions to the augmented problem (4.1), solve linear systems, and predict the positions of singular points, now in terms of the artificial parameter σ, with the methods described in §§ 2 and 3 with $A(\lambda)$ replaced by

$$A(\sigma) \equiv H_y(y(\sigma), \sigma) \equiv \begin{bmatrix} F_x(x(\sigma), \lambda(\sigma)) & F_\lambda(x(\sigma), \lambda(\sigma)) \\ x'(\sigma_0)^T & \lambda'(\sigma_0) \end{bmatrix}. \tag{4.10}$$

Note that now the tangent vector $(x'(\sigma_0), \lambda'(\sigma_0))$ and the eigenvector w (2.7) are always orthogonal, since the last row of $A'(\sigma)$ is identically zero. If we want to switch branches at the bifurcation point, we can use the eigenvector w to predict a regular point on the bifurcating branch.

4.3. Arnoldi iteration on augmented linear systems. The preconditioned Arnoldi iteration described in § 3 can be applied to the augmented problem, if we have a factorization of the bordered matrix $M = A(\sigma_0)$ (4.10). M is sparse since F_x is sparse, so we could factor M directly with a sparse factorization algorithm. However, even if F_x has a sparse factorization, M does not necessarily have a factorization that is just as sparse. This is because if F_x is nearly singular then some pivoting with the last row or column of M is needed for numerical stability when factoring M, which may cause fill-in. If F_x was well conditioned we could factor F_x and apply a block-elimination (see [8]) but this would break down if F_x is singular. In that case we could make a deflated block-elimination as suggested by Chan in [2].

5. Numerical example. In this section we will test the new method on a Taylor problem of hydrodynamics, which is the steady axisymmetric flow of an incompressible viscous fluid between two concentric infinitely long rotating circular cylinders. The inner cylinder is rotating with angular velocity ω_1; the outer cylinder is at rest.

The steady-state Navier–Stokes equations in cylindrical coordinates govern the flow. For any value of ω_1 they admit a stationary solution, the Couette flow, consisting of circular orbits with velocity depending on the distance from the inner cylinder. The Couette flow is stable for small ω_1, but for a critical value of ω_1 the fluid breaks up into horizontal bands called Taylor vortices, and a new fluid motion periodic in the axial direction is superimposed on the Couette flow. When ω_1 is increased still further the Taylor vortices will eventually lose their stability and the fluid will bifurcate into a more complicated time-periodic flow.

5.1. Formulation of the problem. We carefully follow Meyer-Spasche and Keller (see [9], [10]) using the same setup and the same discretization technique.

If R_1 is the radius of the inner cylinder and R_2 is the radius of the outer cylinder, then R_1 is used as length scale and $\omega_1 R_1$ as velocity scale. The relative gap width δ is defined by $\delta \equiv (R_2 - R_1)/R_1$, and the Reynolds number is defined by

$$\mathrm{Re} \equiv \frac{\omega_1 R_1^2}{\nu},$$

where ν is the kinematic viscosity of the fluid. Cylindrical coordinates (r, ϕ, z) are used with respective velocity components (u, v, w) and the pressure is denoted by p. The Navier–Stokes equations for these flows are, in dimensionless form:

$$\begin{aligned} &u_r + \frac{u}{r} + w_z = 0, \\ &\Delta u - \frac{u}{r^2} - p_r = \mathrm{Re}\left[uu_r + wu_z - \frac{v^2}{r} \right], \\ &\Delta v - \frac{v}{r^2} = \mathrm{Re}\left[uv_r + wv_z - \frac{uv}{r} \right], \\ &\Delta w - p_z = \mathrm{Re}[uw_r + ww_z]. \end{aligned} \tag{5.1}$$

Here Δ is the axisymmetric cylindrical Laplacian

$$\Delta \equiv \frac{\partial^2}{\partial r^2} + \frac{1}{r}\frac{\partial}{\partial r} + \frac{\partial^2}{\partial z^2}.$$

The wavelength of the flow is denoted by λ and the wavenumber by k. These parameters are related by $\lambda\delta \equiv 2\pi/k$ and enter into the boundary conditions which require periodicity of period $2\pi/k$. The boundary conditions are:

$$\begin{aligned} &u(1, z) = 0, \quad u(1+\delta, z) = 0, \\ &v(1, z) = 1, \quad v(1+\delta, z) = 0, \quad |z| \geqq 2\pi/k, \\ &w(1, z) = 0, \quad w(1+\delta, z) = 0, \end{aligned} \tag{5.2}$$

$$\begin{aligned} &u(r, -\pi/k) = u(r, \pi/k), \\ &v(r, -\pi/k) = v(r, \pi/k), \qquad 1 \leqq r \leqq 1+\delta; \\ &w(r, -\pi/k) = w(r, \pi/k), \\ &p(r, -\pi/k) = p(r, \pi/k). \end{aligned} \tag{5.3}$$

Making a Fourier expansion in the axial direction and using centered finite differences in the radial direction (see [9]), we obtain a two parameter equation of the form

$$F(x, \lambda, \mathrm{Re}) \equiv T(\lambda)x - \mathrm{Re} f(x, \lambda) - b = 0, \tag{5.4}$$

where x represents the values of the Fourier coefficients at the radial net points, $T(\lambda)$ is a sparse square matrix, $f: \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ is a nonlinear function with a sparse Jacobian, and the vector b is a forcing term due to the boundary conditions. For fixed λ or Re we get a problem of the same form as (1.1).

We have used the same gap width $\delta = 0.3755$ as in [10], and we have concentrated our computations to the region, $0.5 \leqq \lambda \leqq 5.5$, $0 \leqq \mathrm{Re} \leqq 600$. The bifurcation diagram in Fig. 5.1(a) shows the bifurcation from Couette flow to Taylor vortex flow. The numbers in the figure indicate the number of vortices in a period of the bifurcated branch.

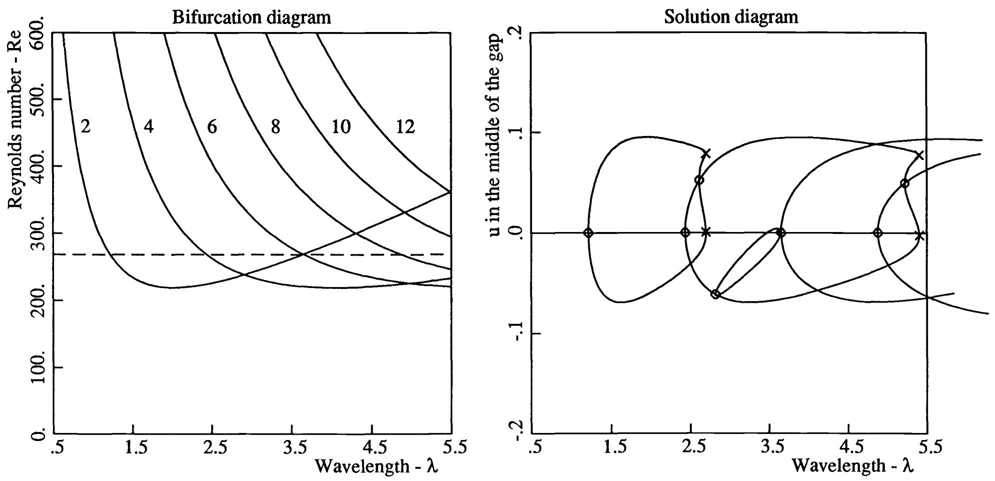


FIG. 5.1. *Bifurcation and solution diagrams.* (a) *Bifurcation from Couette flow to Taylor vortex flow. The numbers indicate the number of vortices in the Taylor flow.* (b) *Solutions along the dashed line in Fig.* (a). *The Taylor solutions form closed loops. Bifurcation points are marked by circles and turning points by crosses.*

In all computations reported here we have used $N=6$ modes in the Fourier expansion and $M=15$ uniformly spaced internal radial gridpoints in the finite-difference approximation. The number of dimensions in our problem then is $n=(2N+1)(2M+1)=403$.

5.2. Implementation details. We have made a FORTRAN implementation of our Algorithm PAC, applied to the augmented problem (4.1).

In step 2(1) of Algorithm PAC, we have chosen to factorize the bordered Jacobian matrix (4.10) directly with the sparse factorization routine MA28 [5].

In step 2(5) of Algorithm PAC we have used the following heuristics to decide when to factorize the Jacobian matrix F_x:

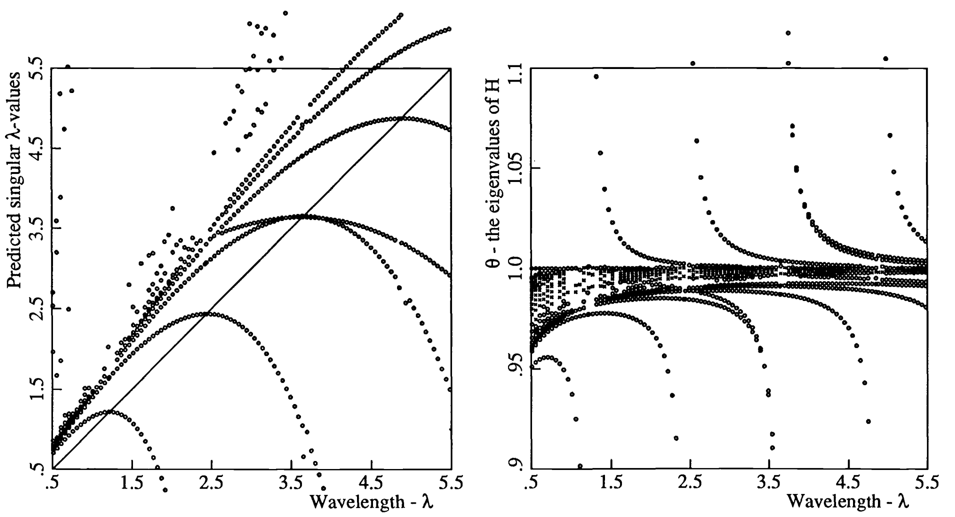


FIG. 5.2. *Prediction of singular points.* (a) *Predicted singular λ-values as functions of the actual λ-value in the continuation.* (b) *The θ-values as functions of λ in the same run. θ passes infinity as λ passes the singular λ-value. Note that when we pass the double singularity at $\lambda=3.6$ two θ-values pass infinity.*

The cost for an iteration of the algorithm PA is essentially a matrix-vector product and a solve; we denote this cost by c. Let n_k denote the total number of iterations with Algorithm PA, when computing the tangent and performing the Newton correction in the kth iteration of Algorithm PAC. The cost of this iteration is then $n_k c$. Furthermore, let f denote the cost of one matrix factorization.

If we had made a factorization of the Jacobian before the kth iteration, the cost would have been $f+\hat{n}_k c$, where we estimate $\hat{n}_k$ by n_0. We decide to factor before the next iteration if $f+n_0 c<n_k c$.

5.3. Continuation in λ—the wavelength. As in [10] we have chosen $\mathrm{Re}=1.5^{1/2}\,\mathrm{Re}_{cr}=268.3$, the dashed line in Fig. 5.1(a), and have calculated the solution diagram using continuation in λ for $0.5\leqq\lambda\leqq 5.5$—this is shown in Fig. 5.1(b). The bifurcation points were located with the technique described in § 4.2 and we switched branches as described in that section. In the figures we represent the solution by $u(1+\delta/2, 0)$, that is, the radial velocity component in the middle of the gap at $z=0$.

If we move along the dashed line of Fig. 5.1(a), with increasing λ, we will first encounter a bifurcation to a two-vortex flow, then a bifurcation to a four-vortex flow,

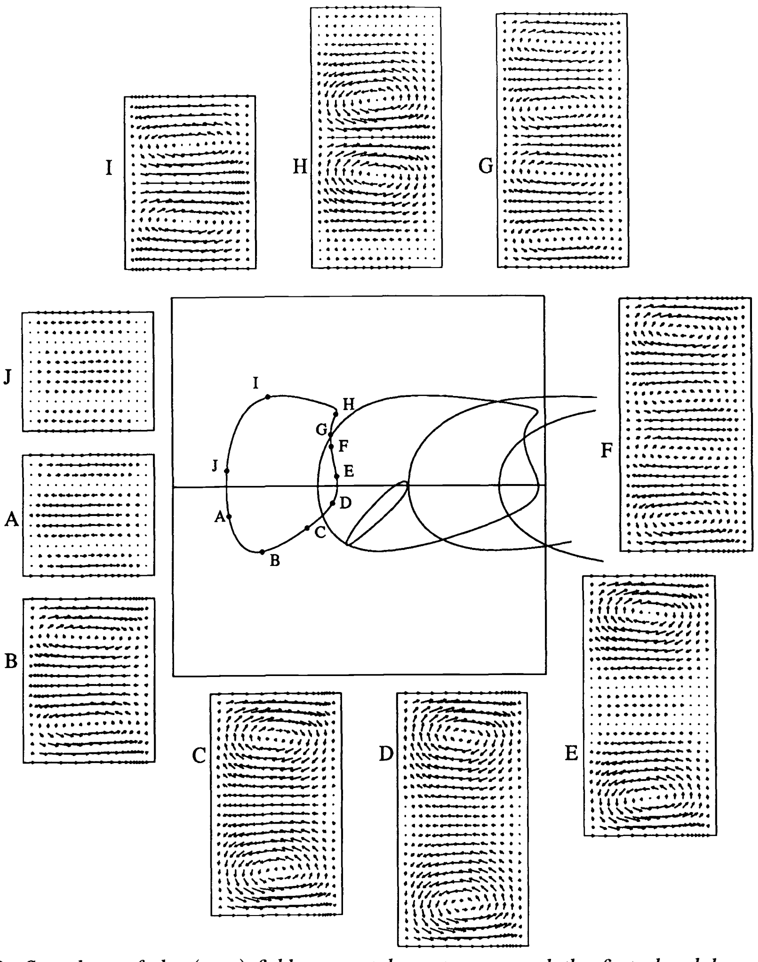


FIG. 5.3. *Snapshots of the (u, w)-field as we take a tour around the first closed loop. If we travel counterclockwise, two new cells are born at the turning point between D and E, and the two old cells die off when passing the turning point near H.*

and finally a multiple bifurcation to both two-vortex and six-vortex flow. As we continue in the direction of increasing λ we will encounter an eight-vortex flow, then a ten-vortex flow, and once again the four-vortex flow, and so on.

In Fig. 5.2(a) we have plotted the predicted singular λ-values as we move along the Couette branch. On the Taylor branches we also get predictions of singularities ahead, but the plot would not look so nice since we have to change parameter directions from time to time. In Fig. 5.2(b) we have plotted the θ-values corresponding to the λ-values of Fig. 5.2(a). As we pass a singular λ-value, the corresponding θ-value passes infinity.

If we move around the first closed loop of Fig. 5.1(b) starting at the bifurcation from the Couette branch, we first pass a turning point, then we pass a secondary bifurcation with the second closed loop, and then we pass a second turning point, and finally we reach the starting point. In Figs. 5.3 and 5.4 we show snapshots of the (u, w)-field and pressure lines during this tour. They show that we started with a two-vortex flow, then two new vortices were formed as we passed the first turning point, giving a four-vortex flow as we pass the bifurcation with the four-vortex flow

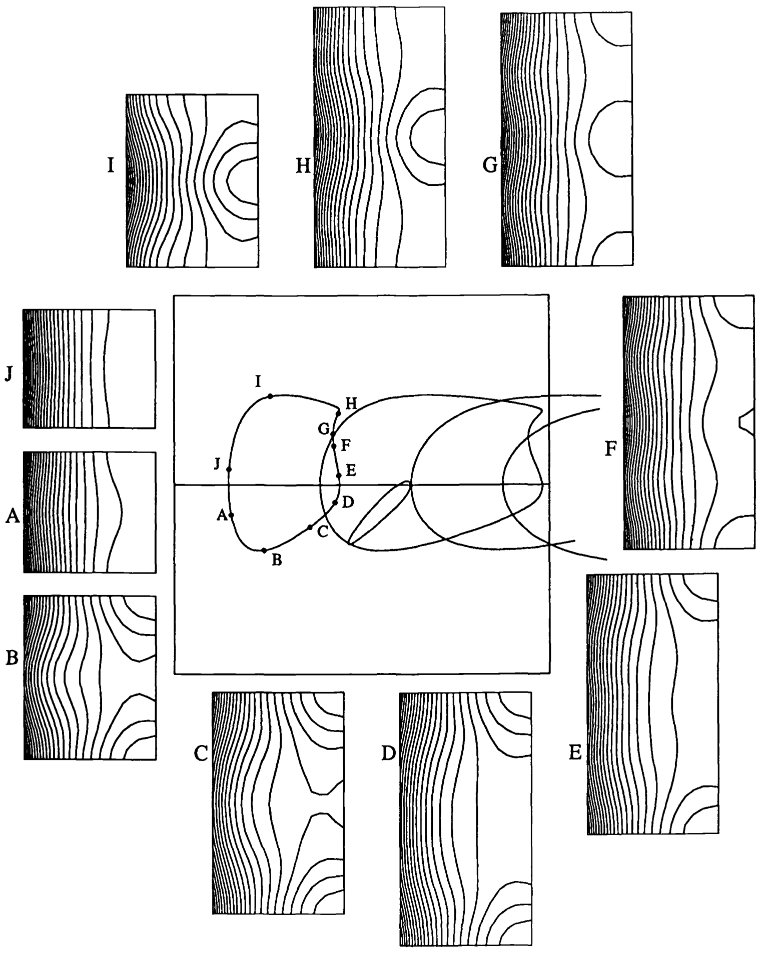


FIG. 5.4. *Snapshots of the pressure lines as we take a tour around the first closed loop. The pressure is highest at the right side of the section, that is, near the outer cylinder, and the pressure levels are equally spaced in the total variation of the pressure.*

of the second loop. The two old vortices die off when we pass the second turning point, giving a two-vortex flow in the opposite direction compared with the two-vortex flow on the lower half of the loop. The snapshots show the right section of the gap over one period in the axial direction.

5.4. Continuation in Re—the Reynolds number. For different values of $\lambda = \lambda_0$ we computed solutions for $0 \leqq \text{Re} \leqq 600$. We predicted singular points and changed branches as for the λ-continuation. The solutions show no closed loops and no secondary bifurcations, at least not in the parameter ranges we covered.

The prediction of singular Re-values along the Couette branch are rather special since the scaling makes the solution constant on the Couette branch $x(\text{Re}) \equiv x_0$. This makes the Jacobian affine in Re:

$$A(\text{Re}) \equiv F_x(x(\text{Re}), \lambda_0, \text{Re}) = T(\lambda_0) - \text{Re}\, f_x(x_0, \lambda_0) = A_0 - \text{Re}\, B_0,$$

where A_0 and B_0 are constant matrices. The linearization then only amounts to a spectral shift. Instead of the curved lines of Fig. 5.2(a) we get straight horizontal lines, that is, we could make a single prediction to get all bifurcation points of (5.4), for fixed $\lambda = \lambda_0$ on the Couette branch.

REFERENCES

[1] P. N. Brown, *A local convergence theory for combined inexact-Newton/finite-difference projection methods*, SIAM J. Numer. Anal., 24 (1987), pp. 407-434.

[2] T. Chan, *Techniques for large sparse systems arising from continuation methods*, in Numerical Methods for Bifurcation Problems, T. Küpper, H. D. Mittelman, and H. Weber, eds., Birkhäuser, Basel, 1984, pp. 116-128.

[3] T. F. Chan and K. R. Jackson, *Nonlinearly preconditioned Krylov subspace methods for discrete Newton algorithms*, SIAM J. Sci. Statist. Comput., 3 (1984), pp. 533-542.

[4] T. F. Chan and Y. Saad, *Iterative methods for solving bordered systems with applications to continuation methods*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 438-451.

[5] I. S. Duff, MA28—*A set of Fortran subroutines for sparse unsymmetric linear equations*, Harwell Report AERE R-8730, Her Majesty's Stationery Office, London, 1977.

[6] H. C. Elman, Y. Saad, and P. E. Saylor, *A hybrid Chebyshev Krylov subspace algorithm for solving nonsymmetric systems of linear equations*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 840-855.

[7] 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.

[8] H. B. Keller, *Numerical solution of bifurcation and nonlinear eigenvalue problems*, in Applications of Bifurcation Theory, P. Rabinowitz, ed., Academic Press, New York, 1977, pp. 359-384.

[9] R. Meyer-Spasche and H. B. Keller, *Computations of the axisymmetric flow between rotating cylinders*, J. Comput. Phys., 35 (1980), pp. 100-109.

[10] ———, *Some bifurcation diagrams for Taylor vortex flows*, Phys. Fluids, 28 (1985), pp. 1248-1252.

[11] W. C. Rheinboldt, *Solution fields of nonlinear equations and continuation methods*, SIAM J. Numer. Anal., 17 (1980), pp. 221-237.

[12] E. Riks, *An incremental approach to the solution of snapping and buckling problems*, Internat. J. Solids and Structures, 15 (1979), pp. 529-551.

[13] A. Ruhe, *Algorithms for the nonlinear eigenvalue problem*, SIAM J. Numer. Anal., 10 (1973), pp. 674-689.

[14] Y. Saad, *Practical use of some Krylov subspace methods for solving indefinite and nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 203-228.

[15] 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.

[16] Y. F. Zhou and A. Ruhe, *Numerical path following and eigenvalue criteria for branch switching*, in Large Scale Eigenvalue Problems, J. Cullum and R. A. Willoughby, eds., North-Holland, Amsterdam, 1986, pp. 121-142.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 6, pp. 1193-1211, November 1990

ANALYZING HIGH-DIMENSIONAL DATA WITH MOTION GRAPHICS*

CATHERINE HURLEY† AND ANDREAS BUJA‡

Abstract. Some new methods for analyzing high-dimensional data, based on real-time graphics, are described. Three-dimensional point cloud rotations provide the canonical example of the applications of motion graphics to data analysis. Similarly, motion may be used to good effect to explore data of arbitrarily high dimension. This will be demonstrated by describing how a data analyst guides a projection plane as it moves through high-dimensional data space.

Key words. high-dimensional data, exploratory data analysis, multivariate analysis, motion graphics, projections, grand tour, guided tour, high-interaction interfaces

AMS(MOS) subject classifications. 6204, 68U05, 6207, 62H25, 62H30

1. Introduction. Graphical methods provide major data analysis tools; they are invaluable for obtaining and presenting information. We use histograms and scatterplots routinely to display univariate and bivariate data. With the limitations of two-dimensional plotting surfaces, effective display of multivariate data presents more of a challenge. In this case, we usually resort to projections of the observations onto one- or two-dimensional subspaces.

Motion graphics is one way of adding power to a display. The illusion of motion is created by displaying new plots in quick succession on the screen. Provided the motion is reasonably smooth, we may visually connect the plots by following points as their position changes throughout the sequence. Therefore, a moving plot contains far more information than a sequence of static plots.

The potential of motion graphics for data analysis has been demonstrated by the PRIM-9 system (Fisherkeller, Friedman, and Tukey (1974)), and more recently, by the PRIM-H (Donoho et al. (1982)) and ORION systems (Friedman, McDonald, and Stuetzle (1982); McDonald (1982)). Commercial systems have become available on affordable hardware (Donoho, Donoho, and Gasko (1985); Parker (1986); Young, Kent, and Kuhfeld (1988)). These programs produce moving scatterplots by projecting observations onto a sequence of two-dimensional subspaces in R^3. The human visual system interprets the result as a rotating three-dimensional point cloud, enabling us to see its full three-dimensional structure.

The subject of this paper is motion graphics for analyzing data of possibly more than three dimensions. More precisely, we consider moving plots obtained by projecting observations onto a sequence of low (one or two) dimensional subspaces in R^p, where p may be arbitrarily large. Of course, except for $p=3$, such moving plots generally cannot display the entire p-dimensional shape of the data. Nevertheless, this approach is fruitful. With motion, several types of low-dimensional structure are easily recognized: heterogeneity such as presence of outliers, and low-dimensional manifolds such as clusters, curves, and two-dimensional surfaces. For an investigation into the power of projection-based methods, see an interesting paper by Furnas (1988).

* Received by the editors May 16, 1988; accepted for publication (in revised form) October 19, 1989. This research was partly supported by the U.S. Department of Energy under contract DE-FG06-85-ER25006 and the National Science Foundation under contract DMS-8504359 to the University of Washington, Seattle, where the authors were with the Department of Statistics.

† Department of Statistics/Computer and Information Systems, The George Washington University, Washington, D.C. 20052.

‡ Bellcore, Morristown, New Jersey 07960.

Effective use of motion demands that the data analyst control the sequence of projections. Therefore, we developed a small but powerful set of tools for constructing *guided tours*—user-controlled sequences of projections. These are described in § 2. Guided tours have been implemented in a program called the *data viewer*[1] (see also Buja et al. (1988); Hurley (1987)). The success of these methods hinges on easily guided motion, leading us to questions of user-interface design. This is discussed in § 3, where we describe a graphical interface and give some practical illustrations with the aid of a few (static) displays. The term "guided tour" was chosen to indicate the closeness of the present work to the "grand tour" (Asimov (1985)) as well as the critical differences between the two concepts (see § 2.1).

2. Projections of high-dimensional data. Suppose the data set consists of p numerical variables observed on each of n cases. Each case may be represented by a p-dimensional observation vector $\mathbf{z}_i$, whose jth element is its value for variable j. The canonical basis vectors in R^p, denoted by $\mathbf{e}_j, j = 1, \cdots, p$, are in one-to-one correspondence with the observed variables.

Consider plots formed by projecting observations from R^p to R^2 or R^1. Typically, we use projections orthogonal with regard to the canonical inner product in R^p, since this allows us to plot pairs of observed variables on perpendicular x and y axes. Note that this inner product assigns unit length to the observed variables, therefore it may be necessary to pretransform them to some standard units, such as zero-mean and unit standard deviation. A two-dimensional projection results in a scatterplot where each point represents a case. Naively, a one-dimensional projection would be displayed as a "dot-plot" consisting of points along a line, but this contains limited information. Instead, we chose to display a marginal density estimate for the projected observations. Moving density plots have previously proven useful in the PRIM-H system (Donoho et al. (1982)) as enhancements of three-dimensional motion graphics.

Moving plots (scatterplots or density plots) are produced by displaying a different projection every fraction of a second. These projections may be characterized by a sequence of planes $P_1, P_2, P_3, \cdots$ in R^p. We now describe a construction for such sequences of planes.

Convention. For brevity, we will use the term *planes* to refer to both one and two-dimensional subspaces.

2.1. Requirements for the sequence of planes. A moving plot should have the following properties which can be expressed as requirements for the sequence of planes:

(1) The plot should move *smoothly*, so that we may observe the changing position of individual points. In practice, a plot appears to move smoothly as long as the position of each point varies smoothly. This is the case if the planes P_i step in small increments along a smooth one-parameter family of planes (Asimov (1985)).

(2) Since our purpose is exploratory data analysis, the moving plot should be under *user control* so the data analyst can pick particular (sequences of) projections.

(3) Plot motion should be performed in *real time*. The alternative is animation, where all computations are performed in advance. The need for real-time motion is a consequence of (2).

From an implementer's point of view it is far easier if the sequence of planes is constructed without allowing human intervention. For example, Asimov (1985) and Buja and Asimov (1985) describe some algorithms for automatically constructing smooth sequences of two-planes. Their *grand tour* sequences are designed to be dense

[1] Implemented on a Symbolics Lisp machine, a single-user graphics workstation.

among all two-planes in R^p, so the moving scatterplot comes close—eventually—to any given two-dimensional projection. However, grand tours suffer from a lack of user control; in dimensions of five or more, there is no guarantee of observing existing structure in a reasonable time period. In six dimensions, it takes a minimum of 6,000,000 planes to get within 10 degrees of all possible planes (Asimov (1985)). At the rate of 10 planes a second, this is 160 hours of viewing time! (In fairness, though, these figures neglect to account for additional information yielded by smooth motion.)

2.2. Constructing the sequence of planes. At the opposite extreme from automatically provided sequences of planes, the data analyst is responsible for choosing sequences where consecutive elements are close. Clearly, this task is far too demanding. A more reasonable, intermediate approach simplifies the analyst's task at the expense of limiting motion control.

Our proposal is to construct sequences of planes by interpolating between consecutive elements of a user-chosen sequence of planes called *target* planes. We refer to the resulting sequences as *guided tours*, because the analyst guides the motion by selecting target planes. Smoothness of motion then depends on the interpolation scheme.

2.2.1. Interpolating between planes. In a similar context, Buja and Asimov (1985) proposed geodesic interpolation paths between pairs of planes, notably for their ideal smoothness properties. Such a geodesic path is generated by rotations in the subspace spanned by the two planes (Wong (1967)).

In the simplest case, the two planes U_0 and U_1 are one-dimensional subspaces, which may be characterized by the unit vectors $\mathbf{a}_0$ and $\mathbf{a}_1$, respectively. Geodesic interpolation is obtained by moving a unit vector $\mathbf{a}$ along the great circle connecting $\mathbf{a}_0$ and $\mathbf{a}_1$. More concretely, let α be the angle between $\mathbf{a}_0$ and $\mathbf{a}_1$, and $\mathbf{a}_1^*$ be the unit vector obtained by orthogonalizing $\mathbf{a}_1$ with regard to $\mathbf{a}_0$ by a Gram-Schmidt step. A geodesic path $U(t)$ from U_0 to U_1 is given by $\mathbf{a}(t) = \cos t\,\mathbf{a}_0 + \sin t\,\mathbf{a}_1^*$, for $0 \leqq t \leqq \alpha$.

Geodesics between general pairs of two-planes U_0 and U_1 are more intricate. To describe them, we first introduce the so-called *principal vectors*: Let $\mathbf{a}_0 \in U_0$, $\mathbf{a}_1 \in U_1$ be unit vectors attaining the smallest angle between U_0 and U_1. Supplement them with vectors $\mathbf{b}_0 \in U_0$ and $\mathbf{b}_1 \in U_1$, so that $(\mathbf{a}_0, \mathbf{b}_0)$, $(\mathbf{a}_1, \mathbf{b}_1)$ form orthonormal bases for U_0 and U_1, respectively. These four vectors are called *principal vectors* for U_0 and U_1, and the angles α (between $\mathbf{a}_0$ and $\mathbf{a}_1$) and β (between $\mathbf{b}_0$ and $\mathbf{b}_1$) are the corresponding *principal angles.* One can show that $\mathbf{a}_0$ and $\mathbf{b}_1$ are orthogonal, and same for $\mathbf{a}_1$ and $\mathbf{b}_0$. Geodesic interpolation between U_0 and U_1 may be described as a one-parameter family of pairs of orthonormal vectors $(\mathbf{a}(t), \mathbf{b}(t))$, where $\mathbf{a}(t)$ moves from $\mathbf{a}_0$ to $\mathbf{a}_1$ along a great circle, and $\mathbf{b}(t)$ similarly from $\mathbf{b}_0$ to $\mathbf{b}_1$. Both vectors move on great circles at constant (but generally unequal) speeds, arriving simultaneously at their respective targets, $\mathbf{a}_1$ and $\mathbf{b}_1$.

The computational steps are as follows: First find the principal directions $\mathbf{a}_0$, $\mathbf{a}_1$, $\mathbf{b}_0$, $\mathbf{b}_1$, and the principal angles α and β.[2] Construct a unit vector $\mathbf{a}_1^*$ by orthogonalizing $\mathbf{a}_1$ with regard to $\mathbf{a}_0$; similarly, construct $\mathbf{b}_1^*$ from $\mathbf{b}_1$ and $\mathbf{b}_0$. A geodesic $U(t)$ from U_0 to U_1 is given by the moving vectors $(\mathbf{a}(t), \mathbf{b}(t))$ where

$$(1) \qquad \mathbf{a}(t) = \cos t\alpha'\mathbf{a}_0 + \sin t\alpha'\mathbf{a}_1^*, \qquad \mathbf{b}(t) = \cos t\beta'\mathbf{b}_0 + \sin t\beta'\mathbf{b}_1^*,$$

[2] Bjorck and Golub (1973) describe a general-purpose method for finding the principal vectors of two subspaces. In the special case where the subspaces have dimension two, the calculation is far simpler: given arbitrary orthonormal bases $(\tilde{\mathbf{a}}_0, \tilde{\mathbf{b}}_0)$ and $(\tilde{\mathbf{a}}_1, \tilde{\mathbf{b}}_1)$ for U_0 and U_1, respectively, we can maximize $\langle \cos\gamma\tilde{\mathbf{a}}_0 + \sin\gamma\tilde{\mathbf{b}}_0, \cos\delta\tilde{\mathbf{b}}_1 + \sin\delta\tilde{\mathbf{b}}_1 \rangle$ by solving a quadratic equation.

$\alpha'=\alpha/(\alpha^2+\beta^2)^{1/2}$ and $\beta'=\beta/(\alpha^2+\beta^2)^{1/2}$. Then $U(t)$, $0 \leqq t \leqq (\alpha^2+\beta^2)^{1/2}$, is a geodesic from U_0 to U_1. Note that the vectors $\mathbf{a}(t)$ and $\mathbf{b}(t)$ rotate at relative rates α/β. The first principal angle α is zero if and only if the two planes have a nontrivial intersection, and so are contained in a three space. In this case, the interpolation amounts to a three-dimensional rotation in the $\mathbf{b}_0$-$\mathbf{b}_1$-plane around the $\mathbf{a}_0=\mathbf{a}_1$ axis.

A few important points should be noted for the proper understanding of this interpolation scheme. First, the vectors $\mathbf{a}(t)$ and $\mathbf{b}(t)$ generally do not correspond to the horizontal and vertical plotting axes on the screen. Changing from one geodesic path to another typically involves a change in principal vectors within the current projection plane. If we were to insist on using the principal vectors as horizontal (x) and vertical (y) screen projections, we would have to subject the current screen to a trivial within-screen rotation before motion along the new geodesic path could be resumed. To avoid this artifact, we use properly rotated principal vectors for horizontal and vertical projection in such a way that continuity is preserved at the point of change from one path to the next:

$$\mathbf{a}_x(t)=\cos\phi\,\mathbf{a}(t)+\sin\phi\,\mathbf{b}(t),$$

$$\mathbf{a}_y(t)=-\sin\phi\,\mathbf{a}(t)+\cos\phi\,\mathbf{b}(t).$$

The angle ϕ for within-screen rotation stays constant for a fixed path and changes at time of changeover in such a way that $\mathbf{a}_x(t)$ and $\mathbf{a}_y(t)$ remain continuous as functions of t.

As a consequence, the same target plane—if approached from two different starting planes—will generally produce two scatterplots which differ by a rotation (and possibly a reflection). Within a target plane, the pair of basis vectors corresponding to the horizontal (x) and vertical (y) axis cannot be prescribed in geodesic interpolation. Given a starting plane and within it the two directions corresponding to the x and y axes, the geodesic interpolation algorithm automatically determines the corresponding x and y directions in the target plane. This is in agreement with the idea that we interpolate **planes** rather than specific pairs of basis vectors of a plane.

2.2.2. Real-time aspects. So far, we have described how a sequence of planes may be constructed by interpolating along geodesic paths between user-chosen target planes. Clearly, once a sequence of target planes has been chosen, it is possible to precompute a denser sequence of interpolating planes. However, we prefer real-time motion to animation for the superior control it offers the data analyst. In principle, control of speed and backward-forward motion are possible with animation, but only real-time controls allow for on-the-spot improvisation—an essential element of data exploration. The principal human interference consists of interrupting the current motion path and selecting a new target, thereby switching to a new motion path starting from the current point. Whenever the data analyst chooses a new target, motion immediately changes direction.

Real-time control over motion speed is also useful. Since the processor of a graphics workstation runs at a constant pace, speed of motion graphics is effectively controlled by a stepsize parameter which specifies the distance between adjacent planes. As a consequence, there is a trade-off between speed and smoothness; the larger the increment, the rougher the motion. A suitable choice for the stepsize parameter depends largely on the time necessary to produce a new plot which in turn depends on processor speed and the number of cases in the data set. Motion graphics systems usually do not attempt to compute reasonable default speeds, but provide interactive control over the stepsize parameter instead.

In practice, sequences of planes are constructed as follows. Let U_0 be the current plane, while U_1 is the current target. $U(\ )$ denotes the interpolating path from U_0 to U_1. Each new plane is formed by stepping an amount δ along $U(\ )$. Following this, the data is reprojected and a new plot appears. If at position d on $U(\ )$, the analyst chooses a new target U_2, motion changes direction, immediately proceeding from $U(d)$ to U_2. Otherwise, motion stops on reaching U_1 and waits until another target is supplied.

2.3. User-chosen planes. That leaves the problem of choosing the target planes. Consider a target plane U, with orthonormal basis $(\mathbf{a}_x, \mathbf{a}_y)$, where the directions $\mathbf{a}_x$, $\mathbf{a}_y$ correspond to the horizontal and vertical plotting axes, respectively. Some obvious choices of planes are:

(i) $\mathbf{a}_x = \mathbf{e}_i$, $\mathbf{a}_y = \mathbf{e}_j$. Projection onto this plane results in a scatterplot of the ith and jth observed variables.

(ii) $\mathbf{a}_x = \mathbf{a}_1$, $\mathbf{a}_y = \mathbf{a}_2$, where the p-vectors $\mathbf{a}_1$ and $\mathbf{a}_2$ are the first two principal component vectors of the data.

Dynamic interpolation of plots will allow viewers to literally *move* between such projections if they are given means to specify target planes.

However, typing in $2p$ numbers per plane requires far too much effort, detracting from the impact of real-time motion. Aside from a few obvious choices, the data analyst may not have much idea what planes to choose. For these reasons, the data analyst should not have to fully specify the target planes. Rather, we propose that she/he *restricts* targets to subspaces of R^p. This is easily achieved by imposing orthogonality constraints on the vectors $\mathbf{a}_x$ and $\mathbf{a}_y$. It remains to produce target planes satisfying the constraints.

2.3.1. Orthogonality constraints. The purpose of orthogonality constraints is to restrict the target planes to useful subspaces of R^p. A sufficient number of constraints will fully determine a target, but usually they confine exploration to subspaces. For the kind of constraints which we propose below, it suffices that they be imposed on the target planes: as long as successive targets satisfy the same orthogonality constraints, geodesic interpolation will guarantee that intermediate planes will also satisfy the constraints.

As before, let a data set of p variables and n cases be given. Again, the raw variables are held in one-to-one correspondence with the canonical basis vectors $\mathbf{e}_j$ in R^p. Linear combinations of the data are called *derived variables.* For generality, we assume that a set of p linearly independent derived variables associated with coefficient vectors $\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_p$ is given. For now, we assume that these coefficient vectors are orthonormal with regard to the canonical scalar product.

As an illustration, we can always obtain a set of derived variables by principal component directions, sorted according to decreasing variance. An analyst may explore a low-dimensional approximation to the data by confining the planes to the space spanned by the first few principal components. In terms of orthogonality constraints, this is achieved by keeping the sequence of target planes orthogonal to the remaining principal components.

The most obvious way of setting up constraints is to classify variables as *active* or *inactive*, where only active variables can have nonzero components in the target projection. However, considerable flexibility is gained by allowing different constraints for the horizontal and vertical projection directions, $\mathbf{a}_x$ and $\mathbf{a}_y$. This implies a further division of active variables into categories which we denote by A, X, or Y. Intuitively, a variable $\mathbf{c}_j$ is in category A (fully active) if both $\mathbf{a}_x$ and $\mathbf{a}_y$ are allowed to have a

$\mathbf{c}_j$-component. The variable $\mathbf{c}_j$ is in category X (horizontally active) when $\mathbf{a}_x$ can have a $\mathbf{c}_j$-component, but $\mathbf{a}_y$ cannot. Category Y (vertically active) variables are similarly defined. Assuming that the $\mathbf{c}_j$'s are orthogonal, this may be formalized as follows:

The variable $\mathbf{c}_j$ is

(i) An *A variable*, if the target is unconstrained with regard to $\mathbf{c}_j$,

(ii) An *X variable*, if $\mathbf{a}_j$ is orthogonal to $\mathbf{c}_j$,

(iii) A *Y variable*, if $\mathbf{a}_x$ is orthogonal to $\mathbf{c}_j$,

(iv) *Inactive*, if the target (both $\mathbf{a}_x$ and $\mathbf{a}_y$) is orthogonal to $\mathbf{c}_j$.

For a given set of variables, there are only a limited number of orthogonality constraints from which to choose. Therefore, constraint selections can be easily made and modified in real time.

Suppose a plot is obtained by projecting observations onto a target satisfying some of the above orthogonality constraints:

• When all active variables are A variables, projection onto the target plane gives a two-dimensional projection from the subspace spanned by the active variables.

• If the active variables consist of q X-variables and s Y-variables, a plane satisfying the constraints yields a special kind of two-dimensional projection termed an *x-y projection.* In an x-y projection, $\mathbf{a}_x$ and $\mathbf{a}_y$ belong to orthogonal subspaces, each spanned by disjoint sets of variables. The class of x-y projections are particularly suitable for predictor-response data (with possibly multiple responses). If $q=s=1$, the constraints determine a unique projection, resulting in a bivariate scatterplot of the Y variable versus the X variable.

• The constraints result in one-dimensional projections when all active variables are X variables. In this way, marginal density estimates for variables and linear combinations become available.

Some constraint combinations should be avoided. For instance, there should be at least two A variables, or one X variable. (In our current implementation, the treatment of X and Y variables is not quite symmetric, because density estimates are drawn on the horizontal axis only.) At any one time, active variables should be either all A variables, or all X and Y variables. Combinations of A variables with either X variables or Y variables are not easily interpretable, and without any obvious applications.

On the side, we note that geodesic interpolation of x-y projections permits computational and conceptual simplifications in comparison to general two-dimensional projections: When both start and target are x-y projections where the horizontal and vertical directions satisfy the same orthogonality constraints, geodesic interpolation boils down to motion along great circles from the x and y start to the respective x and y target. This ensures that the geodesic interpolation will indeed produce a smooth motion to the target plane in its appropriate x-y orientation.

2.3.2. Nonorthogonal constraint vectors. Canonical correlation and discriminant analysis often provide linear combinations (derived variables) with interesting structure. These derived variables could be used to place orthogonality constraints on the sequence of targets. However, unlike principal components, the canonical or discriminant variates are not orthogonal (with regard to the canonical inner product). For example, consider a g group discriminant analysis. Assuming for simplicity that $p \leqq g-1$, this produces linear discriminants $\mathbf{d}_j$, $j=1,\cdots,p$ spanning data space. (If $p>g-1$, additional vectors $\mathbf{d}_g,\cdots,\mathbf{d}_p$ may be constructed.) As a rule, these will not be orthogonal—by definition, projections onto $\mathbf{d}_j$ have unit variance, and projections onto $\mathbf{d}_k$, $\mathbf{d}_j$ are uncorrelated, $1 \leqq j \neq k \leqq p$ (see Mardia, Kent, and Bibby (1982)).

Nonorthogonality causes the following problems:

• Orthogonality constraints become uninterpretable. When the projection directions $\mathbf{a}_x$ and $\mathbf{a}_y$ are orthogonal to the inactive variables, they are not necessarily linear combinations of the active variables alone.

• Two disjoint sets of variables will no longer span orthogonal subspaces, so that x-y projections cannot be formed.

• A projection onto the plane spanned by $\mathbf{d}_1$ and $\mathbf{d}_2$, for example, will not give a bivariate scatter plot of the first two discriminants. But, data analysts routinely examine the results of a discriminant analysis by plotting pairs of discriminants, one against the other, just as for observed variables.

These problems arise because we initially chose to represent cases by their coordinates for the observed variables $\mathbf{z}_i$, $i = 1, \cdots, n$, and to use projections orthogonal with regard to the canonical inner product in R^p. This amounts to declaring that the observed variables are orthonormal. We can get around the difficulty by representing cases in discriminant coordinates, or equivalently, by choosing an inner product for which the $\mathbf{d}_j$ vectors are orthonormal.

The example of discriminant analysis illustrates that we cannot afford to insist on a fixed notion of orthogonality, since this would place unnecessary limitations on the applications of our methods. The obvious solution is to define a new inner product corresponding to each set of variables $\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_p$, so that these directions form an orthonormal basis. Then, by accompanying a change of variables with a change to the associated inner product, the desired effect is obtained.

Many authors (Tukey and Tukey (1981); Friedman (1987)) recommend sphering the data prior to viewing as a means of removing linear structure. Data is sphered by performing a linear transformation, such that the transformed data has identity covariance matrix. The linear discriminants have this property, as do the standardized principal components $\mathbf{a}_j/\sqrt{v_j}$, where $\mathbf{a}_j$ are the principal component directions, and v_j is the variance of the derived variable associated with the vector $\mathbf{a}_j$.

2.4. Sequences of targets. The previous section described how constraints may be used to guide plane selection. To complete our description of guided tours, it remains to describe the construction of target sequences.

Target planes satisfying the selected constraints are easily obtained: Generate a unit vector $\mathbf{a}_x$ which is orthogonal to any inactive or Y variables. Similarly, generate a second vector $\mathbf{a}_y$ orthogonal to both inactive and X variables, orthogonalize it with regard to $\mathbf{a}_x$, and then renormalize. These two vectors form an orthonormal basis for the target plane.

If the two unit vectors are sampled from the uniform distribution on the unit sphere in R^p, we obtain a random two-plane. The disadvantage of using such random planes as targets is that they can depend on the data only in a limited sense, that is, through the constraints. Alternatively, planes may be chosen on the basis of some index that measures some "interesting" feature of the data (Buja, Hurley, and McDonald (1986)). However, with current hardware, producing such planes is too computationally demanding to be performed in real time, except for some trivial applications. By comparison, random planes are quickly obtained, since no data-based calculations are necessary.

Even when relying on random target planes, there is still a need for additional guidance from the data analyst. Since it is the *sequence* of targets which determine the moving plot, the targets could be combined in different ways to form the sequence. We consider the following five *target sequence constructor* schemes, described below.

Suppose U_0 is the current plane and $\cdots U_{-3}, U_{-2}, U_{-1}, U_0$ is the sequence of targets to date. Each new plane $U_j, j \geqq 0$ is constructed randomly, subject to the orthogonality constraints at the time. Then, targets may be arranged as follows:

(i) **Scan:** $U_1, U_2, U_3, \cdots$.
(ii) **Local scan:** $U_1, U_0, U_2, U_0, U_3, \cdots$.
(iii) **Cycle:** $U_1, U_0, U_1, U_0, U_1, \cdots$.
(iv) **Backtrack:** $U_{-1}, U_{-2}, U_{-3}, \cdots, U_{-k}, \cdots, U_{-3}, U_{-2}, U_{-1}, U_0$.
(v) **Rotate:** $U_0 \cdots (U_1)$.

The **scan** scheme results in moving plots formed by interpolating between randomly selected planes $U_1, U_2 \cdots$. In fact, this is an algorithm proposed by Buja and Asimov (1985) for producing a grand tour, and the orthogonality constraints allow for a variety of grand tour like options. When the $q < p$ active variables are A variables, the result is a grand tour restricted to a subspace. In light of the huge numbers of planes required for a "complete" grand tour, this variation is a necessity for its practical application to data sets with more than four variables. If the active variables consist of q X-variables and s Y-variables, we obtain a so-called *correlation tour* (Buja and Asimov (1985)). A correlation tour scans one-dimensional projections of the X variables simultaneously with one-dimensional projections of the Y-variables. It can expose relationships between two groups of variables, providing an exploratory alternative to regression and canonical correlation analysis. Finally, a *one-dimensional tour* is obtained when only X variables are active. If we use density plots to display one-dimensional projections, watching a one-dimensional tour lets us scan the marginal distributions.

Local scan is a variation on **scan**, designed for exploring the "neighborhood" of a plane. Alternate planes are randomly selected from a neighborhood of U_0. A stepsize parameter determines the size of the neighborhood, by specifying how far away from U_0 the random targets can be. By viewing a local scan, the data analyst may establish how the plot changes subject to small changes in the projection plane.

Cycle moves repeatedly along the same path segment by forming a sequence of targets made up solely of two planes. As the projection changes from U_0 to U_1, the viewer mentally connects the sequence of plots, identifying where points in one are located in another. Obtaining this information takes more than one iteration, hence the need for **cycle**.

Backtrack is a limited form of history mechanism, faithful to the path of planes. It is not guaranteed to reconstruct plots exactly as they appeared in the past; for instance, some cases might since have received a new plotting symbol, different color, even been removed from the display. For the simplest case of moving back to the most recent previous target, **cycle** may be used instead of backtrack.

Rotate is somewhat different from the previous four schemes. Typically, the geodesic interpolation produces a rotation towards the user-chosen target, as described in § 2.2. With **rotate**, the data analyst may choose a *particular* rotation in the subspace spanned by a pair of planes, U_0 and U_1, say.

When two planes span a three-dimensional subspace, choosing a rotation is equivalent to choosing an axis of rotation. This allows the data analyst to control the rotation of the three-dimensional point cloud obtained by projecting the observations into the three-dimensional subspace.

In the general case where the planes span a four-dimensional subspace, there are too many possible rotations to choose from. Therefore, we restrict consideration to paths which are similar to the geodesic from U_0 to U_1 in the following sense. Recall that the geodesic from U_0 to U_1 is characterized by a one-parameter family of two orthonormal unit vectors $(\mathbf{a}(t), \mathbf{b}(t))$, where $\mathbf{a}(t)$ and $\mathbf{b}(t)$ rotate at relative speeds α/β.

Then we consider the paths obtained by varying the ratio α/β. Using $\mathbf{a}_0$, $\mathbf{a}_1^*$, $\mathbf{b}_0$, $\mathbf{b}_1^*$ as defined in 2.2 for (1), such paths $\tilde{U}(t)$ are characterized by $(\tilde{\mathbf{a}}(t), \tilde{\mathbf{b}}(t))$,

$$\tilde{\mathbf{a}}(t) = \cos t\theta_a \mathbf{a}_0 + \sin t\theta_a \mathbf{a}_1^*, \qquad \tilde{\mathbf{b}}(t) = \cos t\theta_b \mathbf{b}_0 + \sin t\theta_b \mathbf{b}_1^*, \tag{2}$$

where the ratio θ_a/θ_b is chosen by the analyst. (Compare this to (1).) For example, if $\mathbf{a}_0 = \mathbf{e}_1$, $\mathbf{a}_1 = \mathbf{e}_2$ and $\mathbf{b}_0 = \mathbf{e}_3$, $\mathbf{b}_1 = \mathbf{e}_4$, this allows us to plot any linear combination of variables one and two against any linear combination of variables three and four. If θ_a, θ_b form an irrational ratio, the motion path is dense in the set of all possible such linear combinations. It should be noted that the "target" U_1 is no longer part of such paths, unless θ_a/θ_b is of the form $(\alpha + m\pi)/(\beta + n\pi)$ where m, n are integers.

3. Implementation and interpretation. The previous section described a construction for guided tours—user-controlled sequences of projections. This section describes some elements of an implementation (called *data viewer*) which are crucial to the successful use of these techniques. Unlike other procedures for data analysis, power tools for graphical data exploration come into their own only when embedded in a highly interactive environment. Analytical aspects and human interface questions are hard to separate in an endeavor which tries to bring together the power of dynamic displays and the analytics of multivariate statistics. The goal is to convince the reader that guided tours provide useful and practical tools for analyzing data. Unfortunately, there can be no substitute for actual use. However, we hope to illustrate the potential of these methods by first giving some examples of data viewer displays, and second, describing some aspects of the user interface. The static displays presented here should be thought of as either intermediate steps or destinations of dynamic plot interpolation.

3.1. Display components: Representing cases and variables. Data viewer displays appear in rectangular areas on the screen called *data viewer windows*. Figure 1 shows

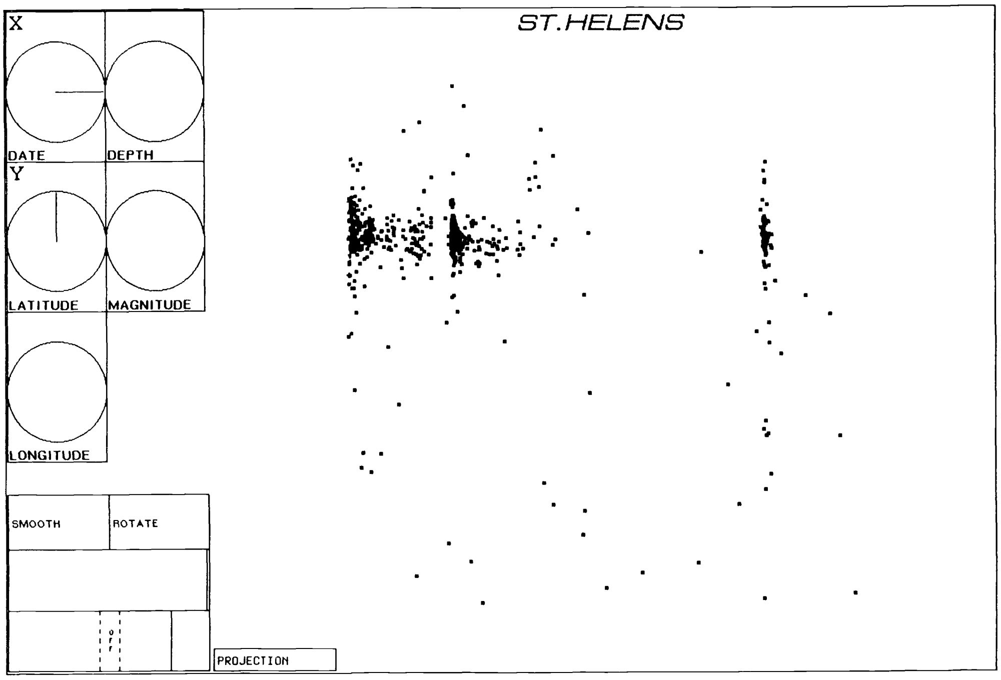


FIG. 1. *A data viewer window.*

a data viewer window for a data set which consists of earthquake data recorded in Mount Saint Helens during May 1980. The central item appearing in a data viewer window is the point scatter, obtained by projecting the observations onto some plane.

An important role in the interpretation of a display is assigned to the *variable boxes* on the left-hand side, where each box represents one variable: The lines or vectors emanating from the box centers identify the current projection plane. More precisely, the line in the box for the jth variable represents the projection of the basis vector $\mathbf{e}_j$ (the unit vector corresponding to the jth variable) onto the current plane. For instance, in Fig. 1 the variable `date` is plotted horizontally while `latitude` is plotted vertically. The projection plane is orthogonal to all variables whose boxes do not show a projection vector. For an example of a less trivial projection see Fig. 3, which is explained below. The projected variable vectors in the variable boxes are the means by which we identify the position of the projection plane in data space since they are a visual equivalent of the projection directions $\mathbf{a}_x$ and $\mathbf{a}_y$. The jth components of $\mathbf{a}_x$ and $\mathbf{a}_y$ are proportional to the horizontal and vertical lengths of the jth projected variable vector. In the language of psychometrics, they represent normalized "factor loadings."

In comparison, it has become customary in three-dimensional graphics to display a three-pod representing the three projected basis vectors. This would not easily generalize to a p-pod for p-dimensional graphics as dealt with in this paper. The clutter would prohibit easy identification of vectors and their association with variables.

Other items appearing in the data viewer display give auxiliary information. For instance, the *variable box labels* drawn in the upper left of the variable boxes informs us about the current orthogonality constraints. A label of X, Y, or A indicates that the variable is currently classified as X, Y, or A, while no label means the variable is inactive. The rectangular region in the lower left is a *control panel.* Its bottom third allows us to control speed and gives a sense of motion via mouse clicks. The top of the panel allows us to select target sequence constructors such as **rotate** or **scan** in the right half, and to toggle between **smoothly** interpolating motion and discontinuous sequences of projections in the left half.

3.1.1. Density plots. For displays of one-dimensional projections, the y coordinates consist of the estimated density at each projected observation, so the density appears as a series of glyphs instead of the usual curve (see Fig. 2). As long as the data set is not too small, such a plot can give a reasonably good picture of the density's shape. Speed is the primary motivation for this plotting technique, since the hardware at hand is not sufficient to display curves moving in real time. In addition, it has the advantage of allowing us to distinguish particular cases, or groups of cases, as the motion proceeds.

As a density estimate we use a *frequency polygon average shifted histogram* (Scott (1985)), which is a proper density estimate contrary to its name. Speed is the main advantage of this procedure over more commonly used kernel density estimators. Like the histogram, it takes time proportional to the number of data points, but avoids the usual lack of smoothness caused by binning. Particularly for motion, it is important that the density estimate be smooth. A slight modification in the one-dimensional projection causes discrete jumps in the histogram bin counts, but only small changes in level for smooth density estimates. Jumps distract us from observing how small changes to the projection affect the marginal distribution. On the other hand, smooth changes enable us to see, for example, how the distribution becomes more skewed as the projection changes.

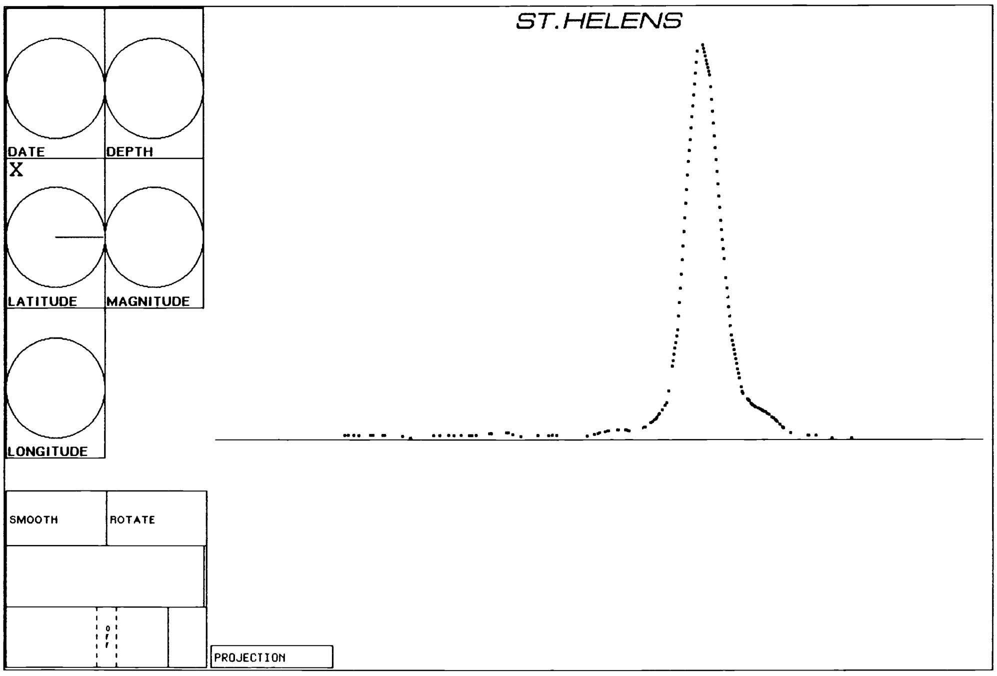


FIG. 2. *A density estimate.*

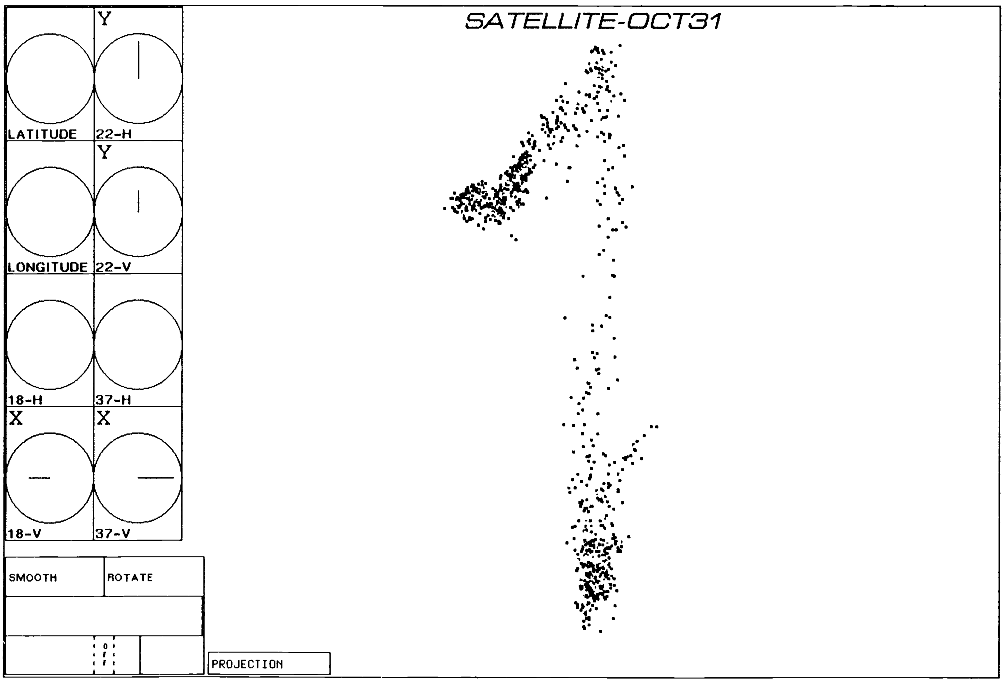


FIG. 3. *Four-dimensional rotations.*

3.1.2. Higher dimensional projections. Figure 3 illustrates a projection from a higher dimensional subspace. The `satellite-Oct31` data consist of microwave brightness temperatures recorded by satellite from a narrow strip stretching from the Bering Sea to the North Pole. The six microwave variables correspond to three frequencies with values for horizontal and vertical polarizations. An examination of the variable box labels and the control panel in Fig. 3 informs us that this projection was obtained by rotating the x-vector $\mathbf{a}_x$ in the space spanned by `18-V` and `37-V`, and simultaneously rotating the y-vector $\mathbf{a}_y$ in the `22-H` and `22-V` plane. The scatterplot has a pronounced "Z"-like shape. By investigating the locations of the rods forming the "Z," we find that the left and right clusters are located at high and low latitudes, respectively, and the remaining points at middle latitudes. In fact, these clusters are the dominant feature of the data.

3.1.3. Derived variables. In the examples so far, the observed variables were used as the current variable set. When derived variables form the current set, we could simply let each variable box represent a derived, rather than an observed variable. Instead, we draw additional boxes on the right-hand side for the alternative set of variables (see Fig. 4). This has a number of advantages:

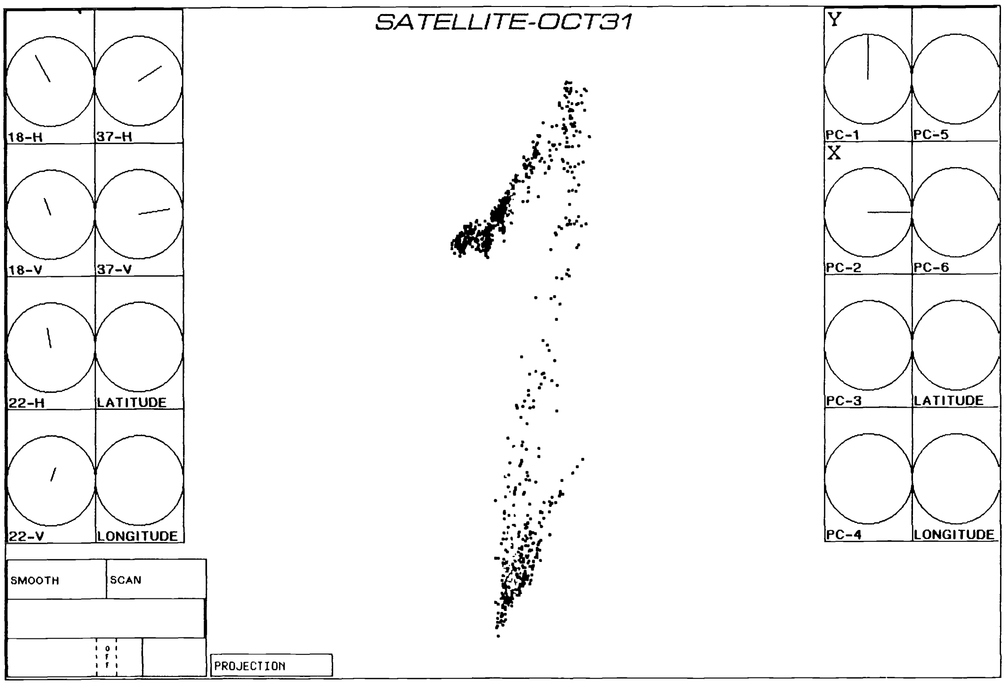


FIG. 4. *Principal components.*

• The user can readily switch from one set of variables to the other, since both are represented in the window.

• More information is displayed: Due to the projected variable vectors drawn in each box, any plot may be interpreted as a linear combination of *either* the observed or derived variables.

A principal components analysis of the microwave readings produced six derived variables, with `latitude` and `longitude` included so that the derived variables span data space. (In this case, it is important that standardization preserves the relative scales of the microwave readings, which all have the same units.) Figure 4 contains a bivariate scatterplot of the first two principal components, displaying a sharper version of the structure evident in Fig. 3. Due to the variance optimizing properties of principal components, Fig. 4 shows a far greater spread of points in the *y* direction, as derived variables are not restandardized prior to plotting. The first two principal component dimensions together account for 99 percent of the variability in the data set, so the remaining principal components will appear as very small blobs in the middle of the plot region. By examining the projected variable vectors in the left-hand side boxes, we see that

(1) The first principal component is a weighted average of the original variables since all variable vectors point upwards;

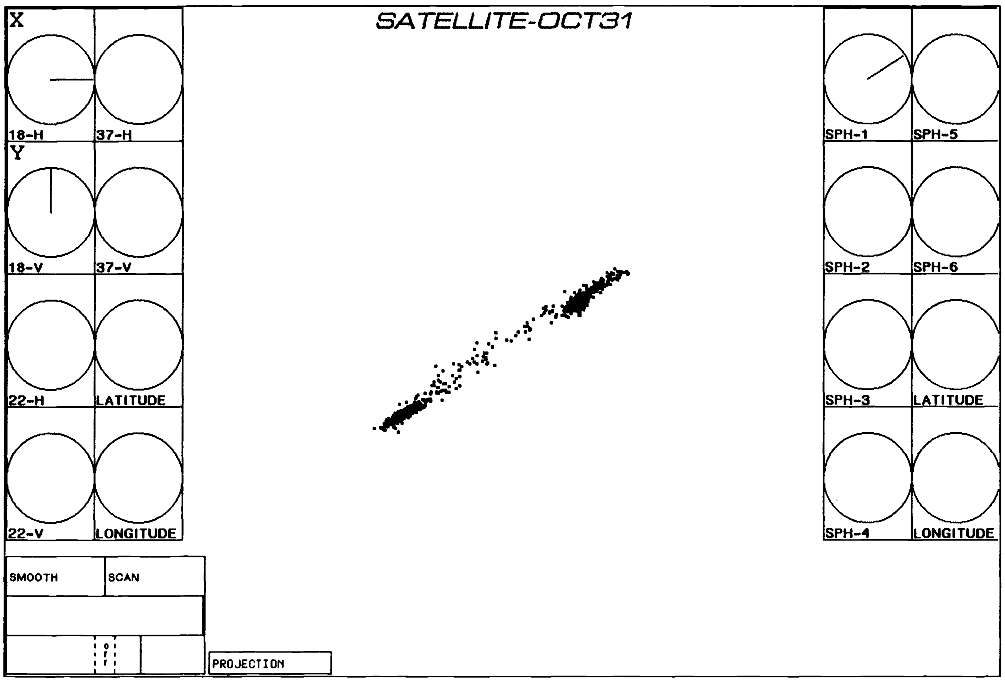
(2) The second principal component contrasts high- and low-frequency measurements since the variable vectors for 18-H and 18-V point left, the ones for 37-H and 37-V point right, while the ones for 22-H and 22-V are more or less vertical.

The first point above is hardly surprising: Microwave readings are strongly influenced by the surface temperature, which we would expect to be the primary source of variability, given the range of latitudes involved. Therefore, it seems as if the first principal component is a surrogate for temperature.

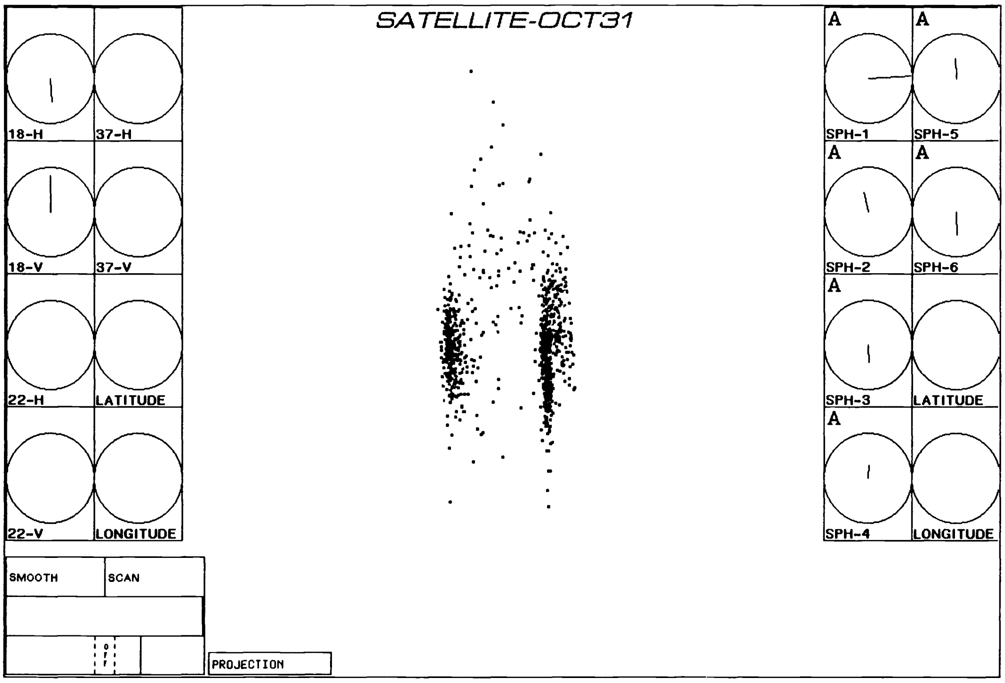
This example demonstrates the analytic interpretation of graphical outputs. More traditionally, principal components are examined by displaying the coefficient vectors in a numerical format. Particularly for large numbers of variables, our graphical representation allows for more immediate, though less precise, interpretation.

The right-hand side boxes in Fig. 5 represent sphered variables, obtained by a standardization of the principal components to unit variance. In Fig. 5(a) we note a (at least initially) surprising effect which derives from restandardization: the current projection onto the variables 18-V and 18-H loads almost exclusively on the first standardized principal component (i.e., the first sphered variable). This implies that the current projection is highly nonorthogonal with respect to the scalar product in standardized principal component space (i.e., sphered coordinates). However, it is immediately clear that this should be so: due to the strong correlation between 18-V and 18-H the direction with small variance (from SE to NW in Fig. 5(a)) will be strongly affected by restandardization to unit variance.

To sphere the scatterplot, we could imagine stretching it in the SE–NW direction. The effect of this manipulation (together with a rotation of about 45 degrees clockwise) is shown in Fig. 5(b), formally produced by a switch to sphered coordinates. Such restandardizations or changes in scale drastically affect perception, as is also documented in Buja et al. (1988). What were the upper right and lower left dense portions of a long narrow point cloud in Fig. 5(a) are two parallel and vertically elongated clusters in Fig. 5(b). The same effects extend to the projected variable vectors: the variables 18-V and 18-H (which still span the projection plane) are no longer orthogonal to each other with regard to the sphered scalar product. Their positions reflect once more the fact that Fig. 5(b) is obtained basically by stretching Fig. 5(a) in the SE–NW direction and rotating it about 45 degrees. In fact, all of these changes in the scatterplot are the result of a single operation, notably the change from observed to sphered coordinates. Of course, geodesic interpolation does not allow for a smooth transition between the plots of Fig. 5, so the second plot appeared instantaneously, with the choice of orientation being made arbitrarily by the system. (An additional operation allows within-screen rotation of projections.) As a side remark, note that variable box labels "A" appear in the right-hand side boxes of Fig. 5(b) indicating that the sphered variables form the current set of variables.



(a)



(b)

FIG. 5. *Sphered coordinates.*

3.2. Designing the user interface. User control was a primary consideration in our development of methods for constructing moving projections. As pointed out, for applications in data analysis, motion should be under immediate user control. At the same time, placing too much of a burden on the user will render guided tour methods inaccessible to all but the most dedicated. We hope to avoid this by providing a small set of motion controls, available to the user through a *graphical interface.*

3.2.1. A graphical interface. In conventional systems for data analysis such as *S* (Becker, Chambers, and Wilks (1988)), Minitab (Ryan, Joiner, and Ryan (1976)), or MATLAB (Moler (1980)), the user types in commands such as `plot(x, y)` to produce a scatterplot. In contrast, the data viewer user operates the program by using a mouse cursor to point at items in a data viewer window, many of which are *mouse sensitive.* Clicks on one of the three mouse buttons cause the display to change. Such a graphical interface is appropriate for applications in real-time graphics, where fast user-program communication is essential.

In § 2.3, we saw that user-guided target selection is the key element in the data viewer's construction of sequences of planes. User-guided target selection is achieved by

(i) Choosing orthogonality constraints, and

(ii) By choosing a target sequence constructor.

The following describes a graphical interface for these user controls. The rectangular area in the window's control panel showing the current choice of target sequence constructor (**rotate, scan,** · · ·) is mouse sensitive. When the mouse cursor moves into this area, a documentation string appears at the bottom of the screen saying `L: Scan, L2: Local Scan, M: Rotate, R: Cycle`. This indicates that, for instance, a double click on the left mouse button changes from `scan` to `local scan`. The symbols `L, L2, M, R` denote left, double left, middle, and right mouse clicks, respectively.

Choosing constraints is somewhat more involved, but is achieved by changing the variable box labels. The circular area marked in each variable box is mouse sensitive and clicks in this region change the labels. As represented in Fig. 6, the mouse cursor's

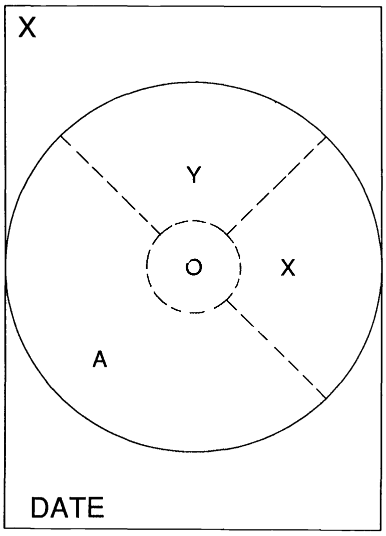


FIG. 6. *Changing shape of the mouse cursor as a function of location within the circular area of a variable box.*

shape changes as it moves through the circular area, where the shape demonstrates the effect of a mouse click. When the cursor shape is X, Y, A, or O, a single click on the left or middle mouse button changes the label to an X, Y, A or blank, respectively. For X and Y cursor shapes, middle clicks have an additional effect. If the box with the mouse cursor receives an X label, then an X in any other variable box is replaced by a blank label. This reduces the number of clicks necessary to pick bivariate scatterplots. Changing the variable box labels usually has no instantaneous effect on the displayed projection, as the labels only affect future target selection. However, smooth motion towards the target is not always of interest, for example, when examining bivariate scatterplots. Therefore, we use double clicks on a mouse button in the variable boxes to mean a change of label *and* a jump to a new target. As a result, mouse clicks in two boxes are sufficient to produce a bivariate scatterplot.

Mouse clicks on the plot region itself toggle motion from off to on. When **rotate** is the target sequence constructor, the position of the mouse cursor specifies a particular rotation within the current four-dimensional subspace. In the degenerate case of a three-dimensional subspace, the points spin around an axis orthogonal to the direction given by the plot region's center and the cursor position. For four-dimensional subspaces, the slope of the line from the mouse cursor to the plot region's center gives the relative rotation speeds θ_a/θ_b (see § 2.4, eq. (2)). In either case, the visual effect is of points moving towards the mouse cursor. For example, note that when the mouse click is near the horizontal line through the plot region's center, the rotation speed for the y direction is comparatively small, so the motion is "almost" a three-dimensional rotation around the y axis.

3.2.2. Consistency of the interface. Ideally, we would like a particular user action to have identical effects in all cases. This makes the system easier to learn and use, because there are no special cases to memorize (Foley and Van Dam (1982, Chap. 6)). As the examples below illustrate, this is not always achievable with our application. At least, the interface should be *consistent,* so that each user action always has a similar effect, whatever the circumstances.

As mentioned previously, clicks in the plot region cause the plot to move. However, when **rotate** is the target sequenceconstructor, these clicks also specify the direction of motion.

The effect of changing constraints depends to some extent on the target sequence constructor. Constructors such as **backtrack** that reuse old targets, simply ignore the orthogonality constraints, whereas schemes like **scan** that provide new targets observe the constraints. Most of the time, label changes do not affect the current projection right away. They affect subsequent motion through immediate selection of a new target which satisfies the new orthogonality constraints, at least in those cases where the target sequence constructor creates new targets.

There are some cases of label changes where one encounters problems with smooth interpolation:

• Switching between two- and one-projections does not allow meaningful interpolation. A discontinuous jump to the target is appropriate and easily anticipated by the user.

• When the start is a general two-dimensional projection, and the target is an x-y projection, a problem in the geometry of geodesic interpolation arises: In this case, geodesic plane interpolation need not arrive at the x-y projection, but a rotated x-y projection. As explained in § 2.2.1, for true plane interpolation the target projection can only be prescribed up to a rotation within the screen, whereas an x-y projection calls for a fixed orientation of the x and y axes up to a sign (unless there is only one x or y axis, in which case even its sign should be prescribed in order to prevent plotting of the negative of a variable). To avoid this conflict, the geodesic interpolation would have to be followed by a trivial within-screen rotation to achieve the desired x-y projection—an option which is not provided in our current system but could easily be implemented.

• A similar case is when start and target planes both give x-y projections, but the start's x (y) subspace overlaps with the target's y (x) subspace. Again, geodesic interpolation would have to be followed by a within-screen rotation.

• An unresolvable case may occur when all but two variables are inactive, but the activated ones change role, e.g., from X, Y to Y, X or A, A to X, Y: in this case, the start and the target would be the same, but they may have opposite orientations. Only a discontinuous reflection or a continuous rotation involving an inactive variable allows us to proceed from start to target.

Other than for certain label changes of variables, a jump (rather than continuous interpolation) is appropriate also when the scalar product changes, as for example, when switching to sphered coordinates. This is indeed one of the more disconcerting transitions for a viewer since rescaling of data space forces a complete rethinking and reinterpretation of perceived structure, as is illustrated by a comparison of Figs. 5(a) and 5(b) (see § 3.1.3).

4. Conclusions.

The goal of this work has been to devise and implement ways of exploring multivariate data based on motion through sequences of projections. The application demands a highly interactive implementation, so that the data analyst controls the exploration process. To summarize, the key issues involved are:

• **Guided tours.** We developed a general paradigm for constructing guided tours of data—namely, by interpolating between consecutive elements of a user-determined sequence of target planes.

• **Choosing target planes.** Rather than precisely specifying each target, orthogonality constraints are imposed which restrict the targets to subspaces. This implies that guided tours include a variety of methods for data exploration, both old and new, from three-dimensional rotations to grand tours on user-defined subspaces.

• **Data-dependent motion.** Arbitrary linear combinations may be used to define new variables. Most usefully, these variables are data derived, for example, through such multivariate methods as principal components, canonical correlations, discriminant analysis, and data sphering. Orthogonality constraints together with data-derived variables allow for data-dependent targets and therefore data-dependent motion paths.

• **The user interface.** A graphical interface is necessary for guided tours. Parameters controlling the motion have graphical representations, so that the user is always aware of their current state. Via mouse interaction, a user can make instantaneous choices in a *direct manipulation* mode. This is particularly important in a system which is capable of exposing a viewer to graphical output in the form of relentless motion.

This paper described a fairly comprehensive set of tools for constructing sequences of projections. But many additions are possible, even within the scope of our paradigm for guided tours. First, we could consider alternative interpolation paths between pairs of planes. More importantly, we could consider further tools for selecting projections. Here, we give brief mention to just a few of the possibilities. Frequently, a variable with a small coefficient in a particular projection (for example, the first principal component) is disregarded when drawing conclusions. For safety, one should examine the plot obtained when the coefficient of this variable is set to zero. Choosing a target by orthogonalizing the current plane with regard to the direction of the variable will achieve this. Conversely, one could choose a target which increases the coefficients of a selected variable, while keeping the relative contributions of the others constant. Additional capabilities could be specially designed for the regression problem. If the best-fitting linear combination and residuals obtained from a user-chosen response and explanatory variables were available at the click of a mouse button, we could produce residual plots, added-variable plots and do interactive stepwise regression. The authors hope to develop methods along these lines in the near future.

Acknowledgments. We thank D. Asimov, G. W. Furnas, J. A. McDonald, and W. Stuetzle for helpful discussions at various stages of this work. We also owe thanks to a referee for invaluable comments and suggestions.

REFERENCES

D. ASIMOV (1985), *The grand tour: a tool for viewing multidimensional data*, SIAM J. Sci. Statist. Comput., 6, pp. 128–143.

R. A. BECKER, J. M. CHAMBERS, AND A. R. WILKS (1988), *The New S Language*, Wadsworth, Belmont, CA.

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

A. BUJA AND D. ASIMOV (1985), *Grand tour methods: an outline*, in Computer Science and Statistics: Proc. 17th Symposium on the Interface, Elsevier, Amsterdam.

A. BUJA, D. ASIMOV, C. HURLEY, AND J. A. MCDONALD (1988), *Elements of a viewing pipeline for data analysis*, in Dynamic Graphics for Statistics, W. S. Cleveland and M. E. McGill, eds., Wadsworth, Belmont, CA.

A. BUJA, C. HURLEY, AND J. A. MCDONALD (1986), *A data viewer for multivariate data*, in Computer Science and Statistics: Proc. 18th Symposium on the Interface, Elsevier, Amsterdam.

D. L. DONOHO, P. J. HUBER, E. RAMOS, AND M. THOMA (1982), *Kinematic display of multivariate data*, in Proc. 3rd Annual Conference and Exposition of the National Computer Graphics Association, Fairfax, VA.

A. W. DONOHO, D. L. DONOHO, AND M. GASKO (1985), MacSpin *Graphical Data Analysis Software*, D^2 Software, Austin, TX.

M. A. FISHERKELLER, J. H. FRIEDMAN, AND J. W. TUKEY (1974), PRIM-9 *An interactive multidimensional data display and analysis system*, in Data: Its Use, Organization, and Management, Association for Computing Machinery, New York.

J. D. FOLEY AND A. VAN DAM (1982), *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, MA.

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

J. H. FRIEDMAN, J. A. MCDONALD, AND W. STUETZLE (1982), *An introduction to real time graphics for analyzing multivariate data*, in Proc. 3rd Annual Conference and Exposition of the National Computer Graphics Association, Fairfax, VA.

G. FURNAS (1988), *Dimensionality constraints on projection and section views of high dimensional loci*, in Computer Science and Statistics: Proc. 20th Symposium on the Interface, American Statistical Association, Alexandria, VA.

C. HURLEY (1987), *The data viewer: a program for graphical data analysis*, Ph.D. thesis and Tech. Report, Department of Statistics, University of Washington, Seattle, WA.

K. V. MARDIA, J. T. KENT, AND J. M. BIBBY (1982), *Multivariate Analysis*, Academic Press, New York.

J. A. MCDONALD (1982), *Interactive graphics for data analysis*, Ph.D. thesis, Stanford University, Stanford, CA.

C. B. MOLER (1980), MATLAB *user's guide*, Tech. Report CS81-1, Department of Computer Science, University of New Mexico, Albuquerque, NM.

D. PARKER (1986), *Acrospin, software running on* IBM PC *compatibles*, ACROBITS, Menlo Park, CA.

T. A. RYAN, B. L. JOINER, AND B. F. RYAN (1976), *Minitab Student Handbook*, Duxbery Press, North Scituate, MA.

D. W. SCOTT (1985), *Average shifted histograms: Effective non-parametric density estimation in several dimensions,* Ann. Statist., 13, pp. 1024–1040.

P. A. TUKEY AND J. W. TUKEY (1981), *Preparation: Pre-chosen sequence of views,* in Interpreting Multivariate Data, V. Barnett, ed., John Wiley, New York.

Y.-C. WONG (1967), *Differential geometry of Grassmann manifolds,* Proc. Nat. Acad. Sci. U.S.A., 57, pp. 589–594.

F. W. YOUNG, D. P. KENT, AND W. F. KUHFELD (1988), *Dynamic graphics for exploring multivariate data,* in Dynamic Graphics for Statistics, W. S. Cleveland and M. E. McGill, eds., Wadsworth, Belmont, CA.

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 6, pp. 1212-1220, November 1990

TIMELY COMMUNICATION

Under the "timely communications" policy for the SIAM Journal on Scientific and Statistical Computing, papers that have significant timely content and do not exceed five pages automatically will be considered for a separate section of the journal with an accelerated reviewing process. It will be possible for the note to appear approximately six months after the date of acceptance.

A DOMAIN DECOMPOSITION ALGORITHM USING A HIERARCHICAL BASIS*

BARRY F. SMITH† AND OLOF B. WIDLUND‡

Abstract. Over the last five years, several new fast algorithms have been developed for the solution of elliptic finite element problems with many degrees of freedom. Among them are Yserentant's hierarchical basis multigrid method and a number of domain decomposition algorithms for problems divided into many subproblems. The condition number of the relevant operators, for the best of these preconditioned conjugate gradient methods, grows only slowly with the size of the problems; the growth typically is quadratic in the logarithm of the number of degrees of freedom associated with a subregion or an element of the coarsest mesh.

Important components of many domain decomposition preconditioners are subproblems related to the sets of variables on the interfaces between neighboring subregions. In this paper, it is demonstrated that, for problems in two dimensions, a very simple change of basis for these one-dimensional problems leads to a preconditioner which is as effective as those previously considered. In the proof, only tools of linear algebra and a result of Yserentant's are used. The numerical experiments confirm that the new algorithm is better conditioned than the original hierarchical basis multigrid method.

Key words. domain decomposition, elliptic equations, finite elements, iterative substructuring, hierarchical bases, preconditioned conjugate gradients, Schur complement

AMS(MOS) subject classifications. 65F10, 65N30

1. Introduction.

We consider second order, self-adjoint, uniformly elliptic differential equations on a two-dimensional polygonal domain Ω. The problems are solved numerically by using continuous, piecewise linear finite elements. The domain is first subdivided into nonoverlapping, triangular subregions Ω_i, also called substructures, and these are further triangulated into elements. H denotes the diameter of a typical substructure and h the diameter of its elements.

We develop a domain decomposition algorithm similar to those considered by Bjørstad and Widlund [4], [5]; Bramble, Pasciak, and Schatz [6], [7]; Dryja and Widlund [15]; and Widlund [25]. When using these methods, the variables interior to individual substructures are first eliminated. The resulting reduced system, the Schur complement, therefore involves only the variables associated with Γ, the set of edges and vertices of the substructures. This system is then solved by a preconditioned conjugate gradient method, where the preconditioner is constructed from certain problems associated with the interfaces $\Gamma_{ij} = \partial\Omega_i \bigcap \partial\Omega_j$ between the substructures, and

* Received by the editors November 27, 1989; accepted for publication (in revised form) April 25, 1990.

† Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, New York 10012. The work of this author was supported in part by the National Science Foundation under grant NSF-CCR-8903003.

‡ Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, New York 10012. This work was supported in part by the National Science Foundation under grant NSF-CCR-8903003 and in part by the U.S. Department of Energy under contract DE-AC02-76ER03077-V at the Courant Mathematics and Computing Laboratory.

a global coarse problem associated with the vertices of the substructures. We note that it is shown in Dryja and Widlund [15] that such a preconditioner naturally can be viewed in terms of a splitting; cf. Varga [24]. In the splitting, the couplings between the groups of variables, associated with individual edges of the substructures, are eliminated. It is also shown in [15] how results and algorithms for the case of two substructures can be used to construct and analyze problems on many substructures.

Various preconditioners have been proposed for the subproblems associated with the edges Γ_{ij}. For each Γ_{ij}, this is essentially a two-subregion problem, and we can therefore take advantage of a number of results obtained in early work on domain decomposition algorithms. Already in 1980, Dryja [13], see also [14], introduced an effective preconditioner J, which is the square root of a discrete, one-dimensional Laplacian on Γ_{ij}. The same preconditioner was also discussed in Bjørstad and Widlund [5] and Bramble, Pasciak, and Schatz [6]. Other preconditioners for these two subregion subproblems, such as the Neumann–Dirichlet algorithm, were considered by Bjørstad and Widlund [5]; Bramble, Pasciak, and Schatz [7]; Chan and Resasco [10]; Chan and Keyes [9]; Dihn, Glowinski, and Périaux [12]; and Golub and Mayers [18]. A number of the resulting algorithms for the many-substructure case are known to be almost optimal in the sense that the condition number is bounded by $C(1+\log(H/h))^2$. For a more complete discussion, see Bjørstad and Widlund [5] and Widlund [25] for the two-subregion and many-subregion cases, respectively.

An alternative almost optimal algorithm, which uses a hierarchical basis, has been introduced by Yserentant [27]. His bound on the condition number is of the same form. The construction of this preconditioner is carried out in two steps. The standard finite element nodal basis is replaced by a hierarchical basis, resulting in a transformed matrix which is much better conditioned. The preconditioner for the transformed matrix is then obtained by discarding the off-diagonal blocks and by replacing all but one of the diagonal blocks by identity (or diagonal, cf. [19]) matrices. The block matrix retained corresponds to the finite element discretization on the coarse mesh defined by the substructures.

In this paper, we consider a hybrid method demonstrating that a successful and simple preconditioner can be obtained by changing the bases of the spaces associated with individual edges Γ_{ij}. Our proof uses only tools of linear algebra and Yserentant's main result. We show that the new method has a smaller condition number than Yserentant's original method; thus it grows no faster than $C(1+\log(H/h))^2$, a result confirmed in our numerical experiments.

Our work has been inspired by the recent works of Babuška et al. [1]; Babuška, Griebel, and Pitkäranta [2]; and Mandel [20], [21], [22], where efficient preconditioners for the p-version of the finite element method are developed by using hierarchical basis functions and partial orthogonalization of the basis functions. Similarly, our algorithm for the h-version involves a change of basis. It is extremely easy to carry out, and it results in a much better conditioned linear system.

We note that an accelerated version of Yserentant's algorithm has been developed by Bank, Dupont, and Yserentant [3]. It can be viewed as a symmetric, successive block overrelaxation method, while the original method is a block Jacobi method. Our hybrid algorithm could similarly be accelerated, but we have not yet implemented such a method. Although we have chosen not to do so in this paper, all the algorithms considered here and in [3] can be viewed as particular instances of additive or multiplicative Schwarz algorithms, cf. Dryja and Widlund [16], [17]. For recent important work on the general multiplicative case, see Bramble et al. [8].

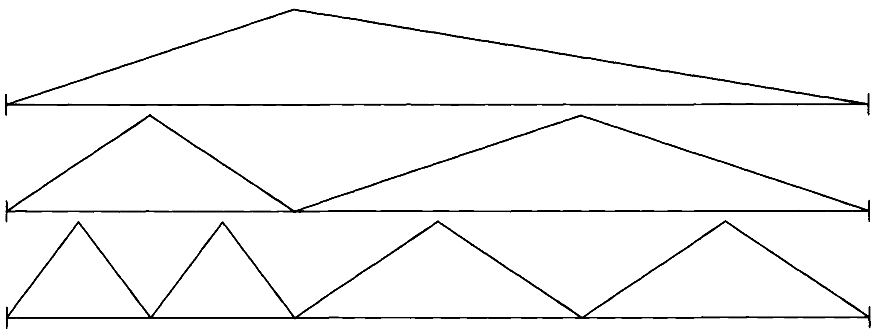


FIG. 1. *Hierarchical basis functions in one dimension.*

2. Yserentant's algorithm and iterative substructuring methods. In this section we provide the necessary background to make it possible to define the new algorithm and to put it in some perspective.

We consider a second order, self-adjoint, uniformly elliptic, bilinear form $a_\Omega(u,v)$ on Ω and, for simplicity, impose a homogeneous Dirichlet condition on $\partial\Omega$:

$$a(u,v) = (f,v), \qquad \forall v \in H_0^1(\Omega), \qquad u \in H_0^1(\Omega).$$

For the two levels of triangulations into substructures Ω_i and elements introduced earlier, we assume shape regularity and the satisfaction of the usual rules of finite element triangulations; see, e.g., Ciarlet [11]. $V^H(\Omega)$ and $V^h(\Omega)$ are the spaces of continuous, piecewise linear functions, on the two triangulations, which vanish on the boundary $\partial\Omega$.

The discrete problem is then of the form

$$a(u_h, v_h) = (f, v_h), \qquad \forall v_h \in V^h(\Omega), \qquad u_h \in V^h(\Omega). \tag{1}$$

2.1. The hierarchical basis method. The hierarchical basis method provides a general-purpose preconditioner for second order elliptic problems in the plane; see Yserentant [26], [27]. The algorithm is given in terms of a set of spaces $V^{h_i}, i = 0, \cdots, j$, which are successive refinements by a factor of two of $V^{h_0} = V^H$. V^{h_i} is the set of piecewise linear finite element functions after i levels of refinement from the original coarse triangulation with $V^{h_j} = V^h$. V^h is a direct sum of subspaces

$$V^h = V_0^h \oplus V_1^h \oplus \cdots \oplus V_j^h,$$

where $V_0^h = V^{h_0}$ and $V_i^h = V^{h_i} \setminus V^{h_{i-1}}$. In other words, V_i^h is the set of piecewise linear functions in V^{h_i}, which are zero at the nodal points of the triangles of all coarser triangulations (see Fig. 1 for a one-dimensional case). For the spaces V_i^h, we choose a basis of standard nodal functions of V^{h_i} associated with the new nodes. The resulting basis for the entire space V^h is much closer to being orthogonal in the H^1 sense than the standard nodal basis functions, and the stiffness matrix is therefore much better conditioned.

Yserentant's preconditioner is a block diagonal matrix in the new basis. The first block is defined by the finite element model for the subspace V^H and the others are diagonal. In matrix notation the resulting system, which is solved by a conjugate gradient method, is of the form

$$D^{-1/2} H^T K H D^{-1/2} \hat{x} = \hat{g}, \tag{2}$$

or equivalently,

$$D^{-1}H^TKH\tilde{x} = \tilde{g}.$$

Here K is the original stiffness matrix, H represents the transformation from the hierarchical to the nodal basis, and D is block diagonal and obtained from H^TKH, as described above. In an implementation, there is no need to represent the stiffness matrix explicitly in the new basis. Instead, we perform the basis change on the vectors as needed.

In [27], Yserentant develops the algorithms needed for performing the basis change from hierarchical basis to nodal basis and back; he also demonstrates that each requires fewer than $2n$ additions and n divisions by 2, where n is the dimension of the finite element space. The following algorithm is valid for both one and two dimensions.

```
Algorithm to form $x \leftarrow Hx$
    for $k = 1$ to number of levels
        for $i$ on level $k$
            $x_i = x_i + (x_{I1_i} + x_{I2_i})/2$
        next $i$
    next $k$
```

The integer arrays $I1$ and $I2$ contain pointers to the two neighbors of i which are on the next coarser level. We can regard the algorithm as defining a factored form of the matrix H. The nodal-to-hierarchical transformation $x \leftarrow H^{-1}x$ is similar. We note that the coarse mesh need not be uniform; see Fig. 1. If the refinements are not uniform, then the weights in the algorithms have to be adjusted. It is also possible to continue the refinement only in selected subregions.

2.2. Iterative substructuring methods. Iterative substructuring algorithms use a different splitting of the space V^h into $N+1$ subspaces;

$$V^h = V^h_{harm} \oplus V^h_0(\Omega_1) \oplus \cdots \oplus V^h_0(\Omega_N) .$$

For each substructure Ω_i, we thus have a subspace $V^h_0(\Omega_i) = V^h \bigcap H^1_0(\Omega_i)$. The elements of V^h_{harm} are piecewise, discrete harmonic functions, i.e., they are orthogonal, in the sense of the bilinear form $a(\cdot,\cdot)$, to all the other subspaces. It is easy to show that an element of V^h_{harm} is uniquely determined by its values on $\Gamma = \bigcup \partial\Omega_i$.

In a first step of many substructuring algorithms, the variables interior to the Ω_i are eliminated. We partition the vector $x = (x_I, x_B)$ and the stiffness matrix K accordingly. The system that remains to be solved is, after a block Gaussian elimination step,

$$Schur(K)x_B = g_B. \tag{3}$$

Here $Schur(K)$ is a Schur complement defined by

$$Schur(K) = K_{BB} - K^T_{IB}K^{-1}_{II}K_{IB}.$$

A particular iterative substructuring method is defined by the choice of a preconditioner for (3). Finally, when accurate enough values on Γ have been computed, the values elsewhere are determined by solving N separate Dirichlet problems on the individual substructures. We note that it is not necessary to compute the elements of $Schur(K)$ since, in the conjugate gradient iteration, this matrix is needed only in terms of matrix-vector products. Such a product can be found at the expense of solving one problem on each of the substructures.

A number of preconditioners can be described as follows: We first carry out a partial change of basis, associating the standard basis functions of V^H with the vertices of the substructures. In the new basis, we represent the Schur complement as

$$\begin{pmatrix} S_{EE} & S_{EV} \\ S_{EV}^T & S_{VV} \end{pmatrix}.$$

Here S_{VV} denotes the part of the Schur complement associated with the vertices of the substructures, S_{EE} the part associated with the edges between substructures and S_{EV} is the part which contains the coupling between the edges and the vertices of the substructures.

The preconditioner for this system is given by

$$\begin{pmatrix} \hat{S}_{EE} & 0 \\ 0 & \hat{S}_{VV} \end{pmatrix},$$

where $\hat{S}_{VV}$ is the coarse mesh finite element problem and $\hat{S}_{EE}$ is a block diagonal matrix. Each of its blocks is associated with the variables of a single edge Γ_{ij}. The operator J, mentioned before, can be used for this purpose; for other examples of such algorithms, see the references given in the third paragraph of §1.

3. The hybrid algorithm. We could combine the two main ideas of §2 as follows: We first represent the stiffness matrix in the hierarchical basis to obtain the system given in (2) and then eliminate the interior variables of all the substructures. We proceed by solving the remaining Schur complement system approximately without further preconditioning. Finally, we use the resulting values as boundary data for the local problems on the individual substructures.

In the new algorithm, we proceed differently, but as we shall see, we will obtain the same approximate solution without the considerable expense of converting the stiffness matrix into the hierarchical basis. In our algorithm we work with the standard nodal basis while eliminating the interior variables, only changing to the hierarchical basis on Γ, the set of interfaces and vertices. The resulting linear system is similar to that of §2.2:

$$D_{BB}^{-1} H_{BB}^T Schur(K) H_{BB} \tilde{x}_B = \tilde{g}_B.$$

It is important to note that we do not use any further preconditioning of the variables associated with the edges Γ_{ij}.

This method offers several possible advantages over the standard hierarchical basis. The conjugate gradient iteration is carried out over a much smaller set of unknowns and we will show that the condition number is smaller. The solution of the subproblems is easily parallelizable since they are independent. The hierarchical basis method in its original form appears to offer less opportunity for this trivial type of parallelization. The change of basis required in each iteration step now consists of completely independent one-dimensional problems instead of a two-dimensional problem. The basic observation is that the values at a node on Γ_{ij} can be computed using only the coefficients for the hierarchical basis functions related to that edge.

We now prove the almost optimality of our algorithm using two simple lemmas and Yserentant's result.

LEMMA 1. *Let G represent a change of basis, which leaves the space of variables on Γ invariant. Then the Schur complement associated with this set of unknowns is independent of the choice of bases for $V_0^h(\Omega_k)$.*

Proof. Let x_I be the vector of unknowns associated with $V_0^h(\Omega_k)$, $\forall\, k$, and x_B be those associated with Γ. The most general basis transformation considered here is of the form

$$\begin{pmatrix} x_I \\ x_B \end{pmatrix} = \begin{pmatrix} G_{II} & G_{IB} \\ 0 & G_{BB} \end{pmatrix} \begin{pmatrix} \tilde{x}_I \\ \tilde{x}_B \end{pmatrix}.$$

In the new basis, the stiffness matrix is

$$\tilde{K} = \begin{pmatrix} G_{II}^T & 0 \\ G_{IB}^T & G_{BB}^T \end{pmatrix} \begin{pmatrix} K_{II} & K_{IB} \\ K_{IB}^T & K_{BB} \end{pmatrix} \begin{pmatrix} G_{II} & G_{IB} \\ 0 & G_{BB} \end{pmatrix}.$$

A straightforward calculation shows that its Schur complement satisfies

$$Schur(\tilde{K}) = G_{BB}^T Schur(K) G_{BB}. \qquad \square$$

The following result follows easily by a Rayleigh quotient argument.

LEMMA 2. *Let K be symmetric, positive definite. Then, the condition numbers of K and its Schur complement satisfy*

$$\kappa(Schur(K)) \leq \kappa(K).$$

Our main result is given in the following theorem.

THEOREM 1. *The condition number of the hybrid algorithm, introduced in this section, is bounded by the corresponding condition number of Yserentant's method. Thus, it is bounded by $C(1+\log(H/h))^2$.*

Proof. We use Lemma 1 twice and the block diagonality of D to obtain

$$\begin{aligned} Schur(D^{-1/2}H^TKHD^{-1/2}) &= D_{BB}^{-1/2} Schur(H^TKH) D_{BB}^{-1/2} \\ &= D_{BB}^{-1/2} H_{BB}^T Schur(K) H_{BB} D_{BB}^{-1/2}\ . \end{aligned}$$

By using Lemma 2, we obtain

$$\begin{aligned} \kappa(D_{BB}^{-1/2} H_{BB}^T Schur(K) H_{BB} D_{BB}^{-1/2}) &= \kappa(Schur(D^{-1/2}H^TKHD^{-1/2})) \\ &\leq \kappa(D^{-1/2}H^TKHD^{-1/2}), \end{aligned}$$

which is bounded by $C(1+\log(H/h))^2$; see Yserentant (Thm. 4.1) [27]. $\square$

4. Numerical experiments. In a first set of experiments, we consider the domain $\bar{\Omega} = \bar{\Omega}_1 \bigcup \bar{\Omega}_2$, where $\bar{\Omega}_1$ and $\bar{\Omega}_2$ are unit squares aligned along an edge $\Gamma = \bar{\Omega}_1 \bigcap \bar{\Omega}_2$. We use the standard regular mesh and the usual five-point discretization for the Laplacian. The results are listed in Table 1. We note that here the values of H/h are $4, 8, \cdots, 256$ and $\log_2(H/h)$ is equal to the number of refinement levels.

Remark. Our experiments show that the condition number grows more quickly than $(1+\log(H/h))$ for the two-subdomain case. We note that for a number of preconditioners the condition number remains bounded in this case. This is true for the preconditioner J if we solve a Dirichlet problem, cf. [5], but not for a Neumann problem. Yet, our method and that based on the J operator both have condition numbers which grow like $(1+\log(H/h))^2$ in the many-subdomain case.

TABLE 1
Condition numbers for the two-subdomain case.

Refinement levels	1	2	3	4	5	6	7
# of unknowns on Γ	3	7	15	31	63	127	255
New method	1.68	2.66	3.82	5.18	6.75	8.52	10.50
No preconditioning	3.05	6.88	14.20	28.63	57.37	114.79	230.49

TABLE 2
Condition numbers for the many-subdomain case.

Refinement levels	3	4	5	6	7	8	9
# of unknowns in Ω	7^2	15^2	31^2	63^2	127^2	255^2	511^2
Yserentant's results	10.59	19.53	31.85	47.14	65.38	86.51	110.49
		4 Subdomains					
# of unknowns on Γ	13	29	61	125			
New method	3.35	5.18	10.87	15.45			
No preconditioning	9.77	21.50	44.97	91.98			
		16 Subdomains					
# of unknowns on Γ		81	177	369	753		
New method		4.89	7.94	11.81	16.45		
No preconditioning		35	75	155	316		
		64 Subdomains					
# of unknowns on Γ			385	833	1729	3521	
New method			5.29	8.52	12.54	17.32	
No preconditioning			137	290	599	1217	
		256 Subdomains					
# of unknowns on Γ				1665	3585	7425	15105
New method				5.46	8.71	12.78	17.61
No preconditioning				546	1152	2372	4766

In a second set of experiments, we consider the case of many substructures. The unit square Ω is subdivided uniformly into 4, 16, 64, or 256 square subdomains and the same model problem is solved using uniform meshes, see Table 2. We compare our results with a set of experiments reported in Yserentant [27]. We note that in his experiments, Yserentant does not solve any linear system corresponding to a coarse mesh. Our experiments are therefore not directly comparable to his.

The values of H/h can easily be computed. Thus if there are 16 subdomains, $H = \frac{1}{4}$ and if the number of unknowns in Ω is 31^2, then $h = 1/32$.

The coarse problem and the problems associated with the edges, which together make up the preconditioner, are independent. We can therefore scale the contribution of the coarse model by a scalar factor α selecting the value of the parameter for which the convergence is fastest. In our numerical experiments, we have found that for our model problem $\alpha \approx 3.6$ is the best for a wide range of refinements. We note that the condition number grows quadratically in the logarithmic factor for any fixed $\alpha > 0$. All of our numerical results are reported for $\alpha = 3.6$.

In other experiments, which will be reported in Smith [23], we have found that there is very little difference in the performance of the new and several previously known domain decomposition methods. We also note that a version of our algorithm has been tested successfully for a membrane model of plane linear elasticity by Anders Hvidsten and Petter Bjørstad of the University of Bergen, Norway.

REFERENCES

[1] I. BABUŠKA, A. CRAIG, J. MANDEL, AND J. PITKÄRANTA, *Efficient preconditioning for the p-version finite element method in two dimensions*, Tech. Report 41098, University of Colorado, Denver, CO, October 1989; SIAM J. Numer. Anal., submitted.

[2] I. BABUŠKA, M. GRIEBEL, AND J. PITKÄRANTA, *The problem of selecting the shape functions for a p-type finite element*, Internat. J. Numer. Methods Engrg., 18 (1989), pp. 1891–1908.

[3] R. E. BANK, T. F. DUPONT, AND H. YSERENTANT, *The hierarchical basis multigrid method*, Numer. Math., 52 (1988), pp. 427–458.

[4] P. E. BJØRSTAD AND O. B. WIDLUND, *Solving elliptic problems on regions partitioned into substructures*, in Elliptic Problem Solvers II, G. Birkhoff and A. Schoenstadt, eds., Academic Press, New York, 1984, pp. 245–256.

[5] ———, *Iterative methods for the solution of elliptic problems on regions partitioned into substructures*, SIAM J. Numer. Anal., 23 (1986), pp. 1093–1120.

[6] J. H. BRAMBLE, J. E. PASCIAK, AND A. H. SCHATZ, *The construction of preconditioners for elliptic problems by substructuring*, I, Math. Comp., 47 (1986), pp. 103–134.

[7] ———, *An iterative method for elliptic problems on regions partitioned into substructures*, Math. Comp., 46 (1986), pp. 361–369.

[8] J. H. BRAMBLE, J. E. PASCIAK, J. WANG, AND J. XU, *Convergence estimates for product iterative methods with applications to domain decomposition and multigrid*, Tech. Report, Cornell University, Ithaca, NY, 1990.

[9] T. F. CHAN AND D. F. KEYES, *Interface preconditioning for domain-decomposed convection-diffusion operators*, Tech. Report, Yale University, New Haven, CT, 1989.

[10] T. F. CHAN AND D. C. RESASCO, *A survey of preconditioners for domain decomposition*, Tech. Report /DCS/RR-414, Yale University, New Haven, CT, 1985.

[11] P. G. CIARLET, *The Finite Element Method for Elliptic Problems*, North-Holland, Amsterdam, 1978.

[12] Q. DIHN, R. GLOWINSKI, AND J. PÉRIAUX, *Solving elliptic problems by domain decomposition methods with applications*, in Elliptic Problem Solvers II, G. Birkhoff and A. Schoenstadt, eds., Academic Press, New York, 1984, pp. 395–426.

[13] M. DRYJA, *An algorithm with a capacitance matrix for a variational-difference scheme*, in Variational-Difference Methods in Mathematical Physics, G. I. Marchuk, ed., USSR Academy of Sciences, Novosibirsk, 1981, pp. 63–73.

[14] ———, *A capacitance matrix method for Dirichlet problem on polygon region*, Numer. Math., 39 (1982), pp. 51–64.

[15] 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, Proceedings of the Conference on Iterative Methods for Large Linear Systems, held in Austin, TX, October 19-21, 1988, to celebrate the sixty-fifth birthday of David M. Young, Jr.

[16] ———, *Towards a unified theory of domain decomposition algorithms for elliptic problems*, in Proceedings of the Third International Symposium on Domain Decomposition Methods for Partial Differential Equations, T. F. Chan, R. Glowinski, J. Periaux, and O. B. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990.

[17] ———, *Multilevel additive methods for elliptic finite element problems*, Tech. Report, Department of Computer Science, Courant Institute, New York, NY, 1990. To appear in the proceedings of a GAMM meeting held in Kiel, Federal Republic of Germany, January 1990.

[18] G. GOLUB AND D. MAYERS, *The use of preconditioning over irregular regions*, in Computing Methods in Applied Sciences and Engineering, VI, R. Glowinski and J. L. Lions, eds., North-Holland, Amsterdam, New York, Oxford, 1984, pp. 3–14. (Proceedings of a conference held in Versailles, France, December 12-16, 1983.)

[19] A. GREENBAUM, C. LI, AND H. Z. CHAO, *Parallelizing preconditioned conjugate gradient algorithms*, Comput. Phys. Comm., 53 (1989), pp. 295-309.

[20] J. MANDEL, *Iterative solvers by substructuring for the p-version finite element method*, Comput. Methods Appl. Mech. Engrg., 1989; to appear in a special issue as Proceedings of an International Conference on Spectral and High Order Methods, Como, Italy, June 1989.

[21] ———, *On block diagonal and Schur complement preconditioning*, Tech. Report, University of Colorado, Denver, CO, November 1989.

[22] J. MANDEL, *Two-level domain decomposition preconditioning for the p-version finite element version in three dimensions*, Internat. J. Numer. Methods Engrg., 1989, to appear.

[23] B. F. SMITH, *Domain Decomposition Algorithms for the Partial Differential Equations of Linear Elasticity*, Ph.D. thesis, Courant Institute, New York, NY, 1990, to appear.

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

[25] O. B. WIDLUND, *Iterative substructuring methods: Algorithms and theory for elliptic problems in the plane*, in First International Symposium on Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988.

[26] H. YSERENTANT, *Hierarchical bases give conjugate gradient type methods a multigrid speed of convergence*, Appl. Math. Comp., 19 (1986), pp. 347–358.

[27] ———, *On the multi-level splitting of finite element spaces*, Numer. Math., 49 (1986), pp. 379–412.